Beherrschen

Sie Ihren Commodore 64®



- ★ Tips und Tricks
- ★ Hochauflösende Grafik
- ★ Tonerzeugung
- ★ Praktische Hinweise
- ★ Viele nützliche Unterprogramme

C. Lorenz

ISBN 3–88963–147–9

Es kann keine Gewähr dafür übernommen werden, daß die in diesem Buche verwendeten Angaben, Schaltungen, Warenbezeichnungen und Warenzeichen, sowie Programmlistings frei von Schutzrechten Dritter sind. Alle Angaben werden nur für Amateurzwecke mitgeteilt. Alle Daten und Vergleichsangaben sind als unverbindliche Hinweise zu verstehen. Sie geben auch keinen Aufschluß über eventuelle Verfügbarkeit oder Liefermöglichkeit. In jedem Falle sind die Unterlagen der Hersteller zur Information heranzuziehen.

1. Auflage 1983

Gedruckt in der Bundesrepublik Deutschland – Printed in West-Germany – Imprime'en RFA.

Dieses Buch ist eine Produktion des HOFACKER Verlages, Ing. W. Hofacker GmbH. Die Produktion erfolgte unabhängig von der Firma Commodore und ihren weltweiten Niederlassungen.

Inhaltsverzeichnis

Grundsätzliches zum Commodore 64 1
Allgemeine Bemerkungen zum Commodore 64. 9
Laden von PET/CBM-Cassetten. 10

BASIC-Programme und TOKENS 11
Änderungen der BASIC-Zeiger 16
Memory Map. 17

Dateien auf Cassette oder Diskette 19
Allgemeines. 19
Daten auf Cassette in BASIC. 19
Speichern und Laden von Datenfiles 20
Wichtige Regeln beim Schreiben auf Band. 24
Lesen von Cassette (LOAD) 24
Der Commodore 64 und die Diskettenstation 1541 28
Allgemeines. 28
Anschluß des Systems 28
Erste Kontakte 29
Weitere interessante Befehle und Funktionen 32
Files auf Diskette 32

Ton und Grafik mit dem Commodore 64 37
Allgemeines. 37
Der VIC-6567 Baustein im C-64 und seine Eigenschaften 38
Kurze Zusammenstellung der wichtigsten Einzelheiten (Blitzkurs) . . 38
Über die Tastatur durch Betätigen der entsprechenden Taste 38
Eine übersichtlichere Darstellung ist die Form PRINT CHR$(x) . . . 38
Farbe durch POKE 39
Die Situation nach dem Einschalten in Adresse 53272 46
Programm zur Verschiebung des Zeichensatzes im RAM. 49
Untersuchung des Zeichengenerators. 50
Programm zur Untersuchung von Zeichen. 51

Der Tonausgabebaustein im Commodore 64 55
Registeraufbau des SID 6581 56
Beispiel für dreistimmige Tonerzeugung 62
Einfache Sound-Beispiele 63
RANDOM Noise 63
Motorengeräusch 64
Musik Stück auf C-64 65
US-Flagge auf C-64 70

Tips und Tricks für die BASIC-Programmierung 73
BASIC-Programme: kürzer und schneller. 73
Umwandlung Hexadezimal – Dezimal. 75
INT mit INTEGER 76
Runden von Dezimal-Zahlen 77
RANDOMIZE-Simulation 78
BASIC-TOKENS und variables Programmieren mit dem C-64. 80
Grundlagen 80
Startadresse des nächsten Blocks. 81
BASIC-Anweisungen 81
Variables Programmieren. 87
Variable Schleifen 87
Variable Variable. 89
Ein bisher nicht denkbares Programm 90
Warten mit Wait 92
Programm Schreibmaschine. 93
Warten auf eine Taste 93
Einfacher Programmschutz 94
PRINT USING 95
Ein einfacher Listschutz 98
Bildschirmausdruck für VC-1515. 99
Stop dem Bildschirm. 100
POKE und der Bildschirmspeicher. 101
PEEKing in den Bildschirm 102
Zeichencodierung 103
Einige wichtige Adressen und deren Funktionen. 105
RESET Knopf für Commodore 64 106
Abspeichern von Variablen auf Diskette 106
Verbesserte Version. 108
Komfortable Bildschirmpositionierung 110
Abfrage von Anwender-Eingaben 110

Druckerformattierung 111
Ausgabeformattierung 112

Hochauflösende Grafik mit dem Commodore 64 115
Entwurf eines kl. Programmes z. Erstellung hochauflösender Grafik . 119
Routine zum Zeichnen eines Punktes 119

Literaturverzeichnis und Quellennachweis. 125

Vorwort

Der Commodore 64 ist vom Konzept her gesehen ein sehr leistungsfähiges Computersystem. Warum, das werden Sie bald selbst verstehen, spätestens jedoch wenn Sie sich eingehender mit dem C-64 beschäftigt haben. Die dazu notwendigen Ideen, Hinweise und Anregungen will ich Ihnen mit diesem Buch geben.

Das von Commodore mit dem C-64 zusammen gelieferte Handbuch führt Sie gut in die Grundlagen ein. Fast alles was Sie in diesem Buche finden baut darauf auf und gibt Ihnen das Handwerkszeug noch tiefer in die Materie einzusteigen.

Neben vielen Tips und Tricks finden Sie auch Vergleiche und Hinweise auf den PET/CBM und VC-20. Dies soll es Ihnen ermöglichen, Programme aus dem Riesenvorrat von CBM-Software zu schöpfen, und diese an Ihren C-64 anzupassen.

Holzkirchen im Herbst 1983 C. Lorenz

In diesem Buch

Bei der Erstellung dieses Buches hat der Verfasser besondere Sorgfalt walten lassen. Alle Programme wurden getestet und haben gearbeitet.

Sollten sich trotzdem Fehler eingeschlichen haben, schreiben Sie uns und geben Sie uns eine genaue Beschreibung Ihres Problems. Aussagen wie "Programm ist defekt" oder "Programm arbeitet nicht" helfen uns in keiner Weise Ihnen weiterzuhelfen. Schildern Sie deshalb den Sachverhalt genau. Oft schleichen sich auch Fehler bei der Eingabe in den Computer ein. Hier empfehlen wir das Auslisten auf einem Drucker mit anschließendem Vergleich mit dem Listing im Buche.

Hat man keinen Drucker, sollte man eine zweite Person zum Vergleichen heranziehen. Einer liest auf dem Bildschirm, der zweite vergleicht mit dem gedruckten LIsting. Unsere Erfahrung hat gezeigt, daß bei einer Fehlersuche auf dem Bildschirm ein Fehler immer wieder übersehen wird.

Zusammenhänge die im mitgelieferten Handbuch beschrieben sind, haben wir praktisch nicht noch einmal erläutert. Ein von Commodore in den USA angebotenes Referenz Handbuch lag uns leider noch nicht vor. Es soll über 400 Seiten haben und sicher wird bald eine deutsche Übersetzung von Commodore erhältlich sein.

Praktisch ließen sich über die vielen Möglichkeiten des C-64 einige zig Bücher verfassen. Man denke nur an den Tongeneratorbaustein, der gegenüber dem AY-8912 und SN 76477 erheblich mehr an Leistung bietet. Aus diesem Grunde haben wir noch drei weitere Bücher in Arbeit, die Sie alle schon vorbestellen können, und bis ca. November 1983 ausgeliefert werden sollen.

1.	Best.-Nr.	146	Hardware Erweiterungen für C-64	39, – DM
2.	Best.-Nr.	124	Programmieren in Maschinensprache	19,80 DM
3.	Best.-Nr.	145	64 Programme für den Commodore 64	39, – DM

Weitere Titel werden folgen.

Übrigens könne Sie die Bücher

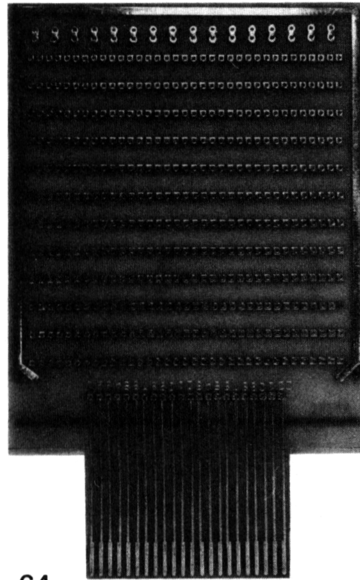
Best.-Nr. 128	Programmieren mit dem CBM	29,80 DM
Best.-Nr. 130	Programmierbeispiele für CBM	19,80 DM
Best.-Nr. 31	57 Praktische BASIC Programme	39,00 DM

auch gut als Programmquellen für Ihren Commodore 64 verwenden. Die Informationen in diesem Buch helfen Ihnen bei den Anpassungsarbeiten erheblich.

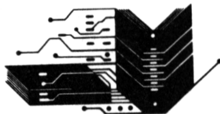
Das System auf welchem wir die Programme aus diesem Buche getestet haben bestand aus:

1 Commodore 64 (gekauft im Mai 1983)
1 Diskettenstation 1541
1 Farbfernsehgerät + 1 Monitor
1 Drucker VC 1515 von Commodore, Device 4 eingestellt.

4963	Editor Assembler, professionell	D	199,–
4965	Professionelle Textverarbeitung CETEXT 64	D	199,–
4970	Externe Experimentierplatine für Erweiterungsport (siehe Bild)		39,–



4980	Adresskartei – 64	C	49,–
4981	Spielepaket – 64	C	49,–
4982	Textverarbeitung – 64	C	99,–
4983	Mini-Assembler – 64	C	49,–
4984	Maschinensprachenmonitor – 64	C	39,80
4985	Disassembler	C	29,80
4986	Busplatine für vier externe Experimentierplatinen (leer)		129,–
4847	Stecker für User Port		19,80
4987	Stecker für Erweiterungsport		19,80



Bücher

145	64 Programme für den Commodore 64	39,–
124	Programmieren in Maschinensprache mit C-64	19,80
146	Hardware Erweiterungen für C-64	39,–

Angebote freibleibend. Zwischenverkauf vorbehalten.

Grundsätzliches zum Commodore 64

Der Commodore 64 kann als eine Art Weiterentwicklung des VC-20 angesehen werden. Er ähnelt jedoch seinem Vorgänger in erster Linie durch seine äußere Erscheinung. Das Konzept innerhalb des Rechners ist jedoch völlig neu.

Ein auf der 6510 CPU basierender Microcomputer mit 64K RAM. Teile des Adressbereiches können wahlweise für RAM- oder ROM-Speicher verwendet werden. Der 6510 ist eine Weiterentwicklung des berühmten 6502 Microprozessors, hat den gleichen Befehlssatz und kann 64K RAM adressieren.

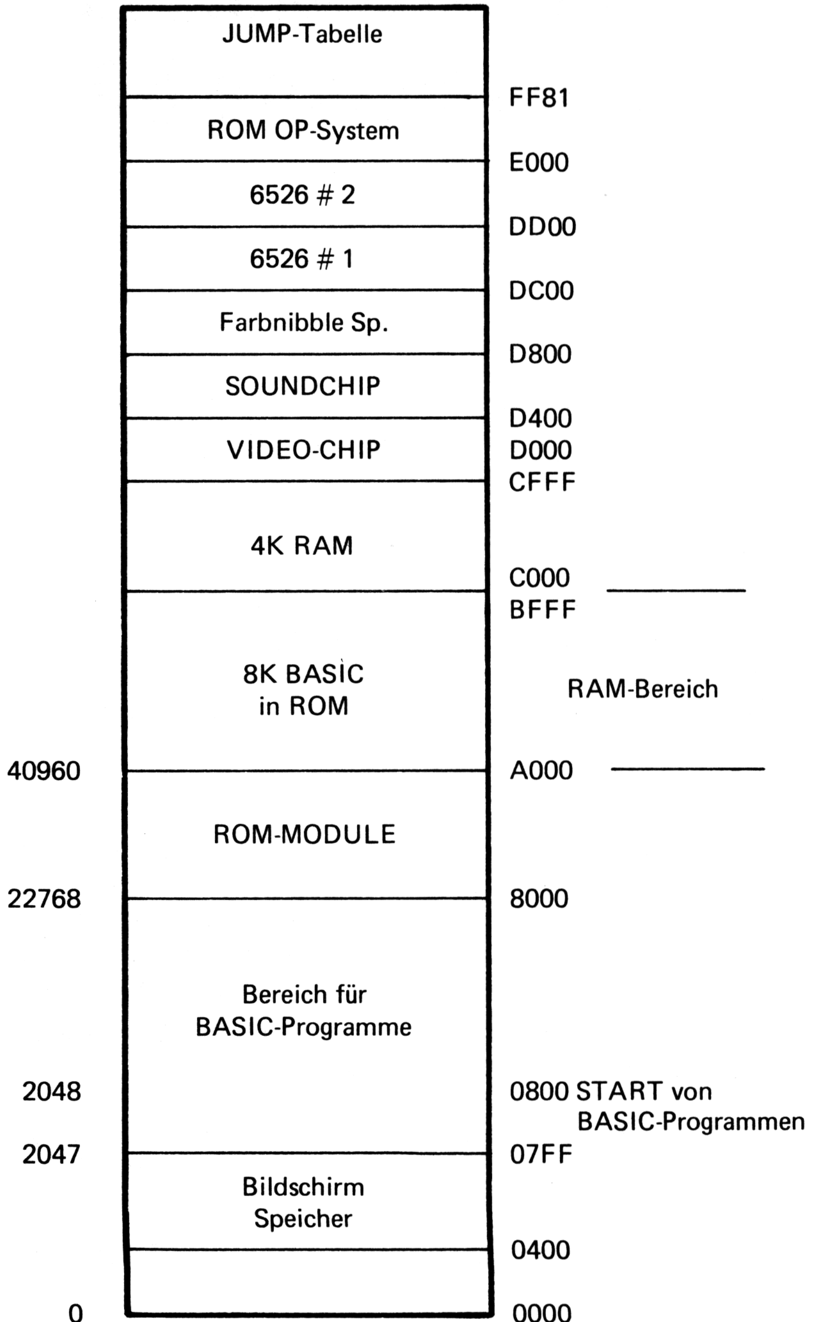
Das Kernel ROM wie wir es vom VC-20 und den anderen Commodore Rechnern her kennen liegt auch hier wieder am oberen Ende des Adressbereiches (E000 – FFFF). Da in diesem Bereich auch RAM vorhanden ist, ist es technisch möglich diesen ROM-Kernel abzuschalten und das darunterliegende RAM mit einem eigenen Kernel zu belegen.

Sie merken schon, was ich damit sagen will. Der Commodore 64 wird damit zu einem recht universellen Computersystem.

Ähnlich können Sie mit dem ROM-BASIC verfahren. Das 8K BASIC im ROM liegt beim Commodore 64 ab A000 Hex bis BFFF.

Die Speicheraufteilung des Commodore 64 finden Sie auf der nächsten Seite.

Speicheraufteilung des Commodore 64



Die Speicheraufteilung kann über bestimmte Erweiterungsportsanschlüsse verändert werden. Die oben gezeigte Konfiguration stellt sich ein, wenn der Erweiterungsport unbeschaltet ist und somit die Anschlüsse

	Pin
ROML	11 = 1
ROMH	B = 1
EYROM	9 = 1
GAME	8 = 1

und CHAREN (kommt von 6510 Pin 28) = 1 sind.

Im Unterschied zu den Rechnern der 2000/3000/4000er Serie

BASIC bei der 2000/3000er Serie ab	C000 Hex
BASIC bei der 4000er Serie ab	B000 Hex
BASIC beim Commodore 64 ab	A000 Hex
BASIC beim VIC-20 (VC-20) ab	C000 Hex

Der BASIC Interpreter des Commodore 64 in ROM ist dem BASIC Interpreter des VC-20 ähnlich. Er liegt verschoben ab Adresse A000.

Auch hier wäre es möglich diesen Interpreter in ROM abzuschalten, und im darunterliegenden RAM-Bereich einen eigenen oder einen abgeänderten Interpreter einzulesen.

Der Commodore 64 ist also eine sehr flexible Maschine und dürfte eine längere Zeit gut verwendbar sein.
Man stelle sich vor, daß man hier neue und leistungsfähigere Sprachen oder sogar aufgabenspezifische Module implementieren kann.

Es ist möglich den Speicherbereich im ROM mit PEEK zu lesen und mit POKE in den dahinterliegenden RAM-Bereich zu bringen. Auf diese Weise können Sie z. B. das BASIC zwischen A000 (4960) und BFFF (49151) von ROM in RAM laden.

Beispiel:

```
10 FOR A = 40960 TO 49141
20 POKE A, PEEK(A)
30 NEXT A
```

Der 6510 bietet gegenüber seinem Vorläufer zusätzlich 6 programmierbare Ein-/Ausgabe-Steuerleitungen, die für unterschiedliche Aufgaben verwendet werden können. Im Commodore 64 werden diese wiefolgt verwendet:

P0 = LORAM
P1 = HIRAM
P2 = CHAREN
P3 = Cassette Schreiben
P4 = Cassette Sense
P5 = Cassetten Motorsteuerung

Diese 6 programmierbaren Ein-/Ausgabe-Ports werden über zwei Register gesteuert, die in den beiden ersten Adressen des Speicherbereiches liegen (0000 Hex und 0001). Die Adresse 0 ist das Datenrichtungsregister, die Adresse 1 ist das eigentliche Kontrollregister.

Wir wollen die Sache hier etwas mehr ausbauen, um Ihnen den grundsätzlichen Aufbau des 64 und den Unterschied zu seinem Vorgänger VC-20 etwas deutlicher zu machen.

Das Datenrichtungsregister an Adresse 0000 Hex sieht wiefolgt aus:

$0000	Ein	Ein	Aus	Ein	Aus	Aus	Aus	Aus	DDRO nach dem Einschalten
			Cassette			D	EF	AB	
$0000			Mot.	Sense	Schr.	ROM	RAM	RAM	Prozessor Port I/O 6510
								LO-RAM	

Recorder Einschalten POKE 1,23
Recorder Ausschalten POKE 1,55

Hierzu unterscheidet sich der Commodore 64 ganz wensentlich von seinen Vorgängern, die an dieser Stelle immer den Anwender USR Sprung Befehl stehen hatten. # 4C = JMP.

In diesen Prozessor Port Register des Commodore 64 können Sie mit POKE Ihre eigenen Werte eingeben und die Speicheraufteilung in gewissen Umfang verändern.

Wie im Speicherbelegungsplan schon angedeutet, nimmt der Commo-

dore 64 nach dem Einschalten eine im Betriebssystem festgelegte Grundstellung ein. Diese Grundstellung ist von den Werten im Betriebssystem und von außen über den Erweiterungsport angelegten Signalen abhängig.

LORAM HIRAM CHAREN	vom 6510 her
EYROM CHAREN	vom Erweiterungsport her

Die oben genannten Signale, sowie weitere zur Verwaltung des mehrfach nutzbaren Speicherbereiches notwendige Informationen, werden im Baustein 82S100 zu einem PLA Schaltkreis zusammengeführt, verknüpft und die notwendigen Steuersignale erzeugt. Dieser Baustein ist die zentrale Verwaltungsstelle für die Speicherorganisation des Commodore 64. Sie erkennen jetzt sicher, welche Möglichkeiten der Commodore 64 von dieser Seite her bietet.

So nun zu unserem ersten Beispiel, welches die Überlagerung von RAM und ROM verdeutlichen sollte.

Nachdem Sie unser Programm zum Transfer des BASIC-Interpreters gestartet haben, dauert es eine kleine Weile und wir sehen READY.

Jetzt muß der ROM-Bereich abgeschaltet werden. Dies kann über den Prozessor Port geschehen. In den Zellen 0 und 1 steht nach dem Einschalten:

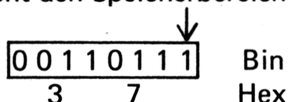
DDR = 47 dez. = 2F hex.
Prozessor I/O Port = 55 dez. = 37 hex.

Dies bedeutet daß das | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | DDR

auf 47 = 2F = 0 0 1 0 1 1 1 1 gesetzt ist und einen 0 Eingang und einen 1 Ausgang bedeutet. (Ähnlich wie beim 6520 und 6522.)

Der Prozessor I/O-Port steht auf 55 dezimal 37 Hex und legt mit

seinem binären Äquivalent den Speicherbereich wiefolgt fest:



Vergleichen wir diese Werte mit unserem vorher gezeigten Registeraufbau so sehen wir daß die ersten beiden Bits Null sind und die drei letzten Register, die den Speicherbereich festlegen jeweils eine "1" enthalten (von links nach rechts gelesen).

Das Bit am äußeren rechten Ende (Bit 0) legt fest, ob der Speicherbereich A000/B000 RAM oder ROM sein soll. Eine Null in diese Adresse gepoked, sperrt das BASIC-ROM und erlaubt uns diesen Bereich über das daraunterliegende RAM zu nutzen. Wir setzen es auf Null, indem wir den Hex-Wert 36 einpoken. Hex 36 = 54 Dez.

POKE 1,54 schaltet nun von einem ROM-BASIC auf ein RAM-BASIC um. Wir sind jetzt in der Lage dieses BASIC zu verändern und unseren eigenen Wünschen anzupassen. Es wäre auch möglich in diesen Bereich ein anderes BASIC von Disk oder Cassette her zu laden und zu verwenden. Ähnlich können wir auch mit dem Kernel und dem Zeichensatz ROM verfahren. Achtung hier kann jedoch das System leicht "abstürzen". Wenn Sie mehr Erfahrung gesammelt haben können Sie auch hier viele Experimente anstellen.

Der Bereich ab E000 – FFFF kann ähnlich verändert werden. Hier müsste man dann diesen Bereich in RAM laden und über POKE 1,53 den ROM-Bereich ab E000 Hex abschalten. Jetzt könnten Sie selbst den Kernel verändern und eventuell sogar einen anderen Prozessor von Ihrem Commodore 64 aus betreiben. Dieser Prozessor könnte über den Erweiterungsport angeschlossen werden und über ein eigenes Monitorprogramm in diesem Bereich gesteuert werden. Man könnte diese Gedanken jetzt noch weiter ausbauen und den I/O-Bereich ab D000 auch noch abschalten (POKE 1,48). Jetzt würde der gesamte Speicherbereich z. B. für eihe Z80 CPU Zusatzkarte zur Verfügung stehen.

Wie schon angedeutet lässt sich die Speicheraufteilung auch noch über Leitungen am Erweiterungsport einstellen. Hier wird abgefragt

ob ein Spiele-ROM oder externes ROM oder ähnliche externe Systeme angesteckt sind.

Der Speicherverwaltungsbaustein ist auch an der Aufteilung beteiligt. Er steuert z. B. über CHAROM das ROM mit dem eingebauten Zeichensatz (U5 2332A von $D000 – $DFFF). Mit dem Zeichensatz ROM könnten Sie ähnlich wie mit dem BASIC- und Kernel-ROM verfahren. In RAM laden und abändern (z. B. Umlaute usw.).

Dies soll zuerst einmal genügen, um Ihnen den groben Aufbau des Commodore 64 zu erläutern.

Der Commodore 64 besitzt weiterhin noch recht gute Hardware zur Ton-, Farb- und Bewegungserzeugung. Die in den ersten Maschinen mitgelieferte Software ist hierfür keinesfalls ausreichend.

Wir werden also bald wieder neue ROM- und BASIC-Versionen angeboten bekommen. Wie wir dies von Commodore ja seit Anfang an gewohnt sind. Hoffentlich ist nicht wieder alles mit allem unkompatibel.

Weitere Interface Bausteine des Commodore 64

1. 6566 Video Baustein (D000 – D02E)
2. 2 x 6526 CIA Baustein (DC00 – DD0F)
3. 6581 SID Ton-Baustein (D400 – D41C)

Die Register dieser einzelnen Bausteine sind, wie wir das beim PET/CBM her kennen auch bestimmten Speicherbereichen zugeordent. Die Programmierung der Bausteine erfolgt durch Verändern der Registerinhalte. Von BASIC her über PEEK und POKE oder auch direkt über den Akkumulator in Maschinensprache. Wir wollen hoffen, daß bald eine BASIC-Version vorhanden ist, die es erlaubt über SET, DRAW und SOUND Befehle, diese Bausteine auf einfache Weise zu bedienen.

Wichtige Adressen

Eine Tabelle mit allen ZEROPAGE, ROM, RAM, I/O und Kernel Adressen ist in jedem Falle von Vorteil.

Zu Beginn wollen wir jedoch die wichtigsten uns am meisten inter-

essierenden Adressen der Übersicht halber zusammenstellen und evtl. Unterschiede zu PET / CBM / VC-20 herausstellen.
Die Zeropage, d. h. die ersten 255 Bytes im Speicher (0 – FF) sind beim Commodore 64 fast identisch wie beim VC-20 Rechner belegt. Manche Adressen sind gleich geblieben, manche haben sich um einige Adressen verschoben. Die beiden ersten Zellen werden anstelle durch den USR-Sprung Befehl 4C30D1 beim PET 2000 durch Register des I/O Prozessor Ports belegt.

Hier eine kurze Übersicht (Dezimal)

Belegung	**Commodore 64**	**VC-20**	**BASIC 1 2000er**	**BASIC 2 3000/4000**
Zeiger: Beginn von BASIC	43–44	43–44	122–123	40–41
Zeiger: Beginn der Variablen	45–46	45–46	124–125	42–43
Zeiger: Beginn der Felder + Arrays	47–48	47–48	126–127	44–45
Zeiger: Ende der Felder	49–50	49–50	128–129	46–47
Zeiger: String Anfang	51–52	51–52	130–131	48–49
Zeiger auf RAM Ende in BASIC	55–56	55–56	134–135	52–53
BASIC Eingabepuffer	512–600	512–600	10–89	512–591
Cassettenpuffer	828–1019	828–1019	826–1017	826–1017
Bildschirmspeicher	1024–2047	7680–8191	32768–36863	32768–36863
Start der BASIC Programme	2048	4096 (1024)	1024	1024
ROM-BASIC	40960–49151	49152–57343	49152–57343	45056–53247
Zeichengenerator ROM	53248–57343	32768–36863		
Tastatur-Puffer	631–640	631–640	527–536	623–632
Anzahl der Zeichen im Tastatur-Puffer	198	198	525	158

Auch die zweite Page ähnelt dem VC-20 sehr. Wer also eine Belegung sucht, kann in seinen VC-20 Unterlagen nachsehen.

Der RAM-Bereich, wo Ihr BASIC Programm anfängt, beginnt beim Commodore 64 an der Adresse 800 Hex.

Allgemeine Bemerkungen zum Commodore 64

Der Commodore 64 ist in jedem Falle ein ausgezeichnetes System. Er wird von der Hardware her gesehen, sicher noch lange auf dem Markt bleiben. Der Systemaufbau erlaubt fast jede denkbare Softwareimplementation. Der Preis wird sich sicher weiter nach unten bewegen.

Folgende Dinge sind mir jedoch unverständlich! Warum hat Commodore wieder einen Rechner ohne RESET-Knopf produziert ? Warum hat man ein BASIC gewählt, welches überhaupt nicht auf die Ton und Graphikeigenschaften eingeht ? Obwohl Commodore den Rechner als Small Business System der unteren Kategorie ansieht und eine CP/M-Karte später anbieten will, ist man bei 40 Zeichen pro Zeile geblieben. 40/80 wäre sinnvoller gewesen. Das Cassettensystem ist wie bei der Commodore CBM 2000/3000/4000 Serie. PET-Programme können in den Commodore 64 eingeladen werden.

Der VC 1515 Drucker mit serieller Schnittstelle kann angeschlossen werden. Wenn die Single-Disk angeschlossen wird, muß der Drucker an die Diskettenstation angeschlossen werden.

Die Tastatur ist sehr gut. Der eingebaute Modulator erlaubt den direkten Anschluß an ein Fernsehgerät. Über den Video-Ausgang kann ein Monitor angeschlossen werden. Dies dürfte gerade bei Anwendungen im Kleinbetrieb von Vorteil sein.

Der Commodore 64 dürfte in Kürze neben dem ATARI und Sinclair Rechner zu den meistverkauften Systemen weltweit gehören. Ich bin sicher daß sich eine solche Maschine bei großen Stückzahlen auch sehr günstig am Markt anbieten lässt.

Laden von PET/CBM-Cassetten

Beim Laden von Cassetten verhält sich der Commodore 64 anders als seine Vorgänger. Nach Load, erfolgt die Anzeige PRESS PLAY. Ist die PLAY-Taste gedrückt, wird der Bildschirm mit der Hintergrundfarbe ausgefüllt. Der Recorder läuft dann, bis er den Namen des nächsten BASIC-Programmes gefunden hat. Jetzt kann die Commodore-Taste rechts am Tastenfeld gedrückt werden und das Programm vollständig geladen werden.

Beim Laden von PET/CBM-Cassetten die für einen BASIC-RAM-Bereich ab Adresse 400 Hex abgespeichert wurden muß man wiefolgt vorgehen:

1. Commodore 64 abschalten und wieder einschalten
2. Laden wie oben angegeben
3. Zuerst Starten und die in der Fehlermeldung erscheinende Zeile löschen.
4. Erst dann listen.

Bemerkung:
Nach dem Laden listen. Oft sieht man am Anfang einige Zahlen, die als Zeilenzahl interpretiert werden. Löschen Sie diese Zahlen und starten Sie das Programm.
PEEK, POKE und SYS Befehle müssen vorher sorgfältig identifiziert und abgeändert werden.

BASIC-Programme und TOKENS

Der Commodore 64 legt die BASIC-Programme ab der Adresse $800 Hex aufwärts im RAM ab. Im Gegensatz zu den Commodore Rechnern der 2000/3000/4000er Serie desen nutzbarer BASIC-Bereich bei $0400 Hex beginnt. Der VC-20 hatte je nach RAM Speichererweiterung den Anfang des BASIC Speichers Bei $1000 Hex, $400 Hex oder $1200 Hex.

Wer also BASIC Programme anpassen oder umschreiben will, sollte sich ein wenig mit dem Aufbau der BASIC-Programme im Speicher befassen. Wir wollen hier in erster Linie auf den Commodore 64 eingehen, wie er Anfang 1983 ausgeliefert wurde. Start des BASIC bei $0800 Hex oder 2048 dezimal.

Das von Ihnen in Ihren Commodore 64 eingegebene BASIC-Programm wird Zeile für Zeile in einem besonders hierfür reservierten RAM-Bereich abgelegt. Der RAM-Bereich wird durch einen Pointer (Zeiger, der auf diese Adresse zeigt) in der ersten Page (1. 255 Byte) des RAM Bereiches festgelegt. Der Zeiger wird beim Einschalten vom Kernal her auf einen bestimmten Wert festgelegt. Beim Commodore 64 sind dies die Zellen 43 und 44 dezimal (2B und 2C Hex).

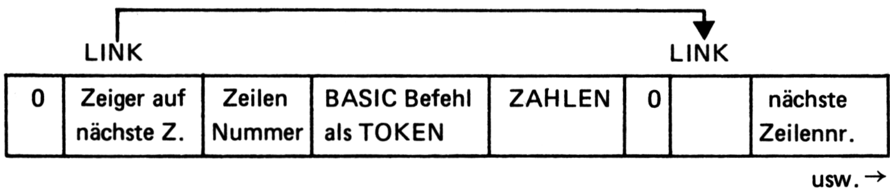
Durch ?PEEK(43) und ?PEEK(44) können Sie sich diese einmal ansehen. Der Inhalt von 43 ist 01 (lower Byte) dez und der Inhalt von 44 (higher Byte) ist 8. Dies bedeutet. Adresse 1 in der 8. Page oder $0801 Hex oder 8 x 256 + 1 = 2049 dez. Das eigentliche BASIC-Programm beginnt also in Adresse 2049 dez. da in 2048 immer eine Null steht.

Lower Byte = niederwertiges Byte
Higher Byte = höherwertiges Byte

BASIC TOKENS

128	END	167	THEN
129	FOR	168	NOT
130	NEXT	169	STEP
131	DATA	170	+
132	INPUT #	171	–
133	INPUT	172	.
134	DIM	173	/
135	READ	174	↑
136	LET	175	AND
137	GOTO	176	OR
138	RUN	177	〉
139	IF	178	=
140	RESTORE	179	〈
141	GOSUB	180	SGN
142	RETURN	181	INT
143	REM	182	ABS
144	STOP	183	USR
145	ON	184	FRE
146	WAIT	185	POS
147	LOAD	186	SQR
148	SAVE	187	RND
149	VERIFY	188	LOG
150	DEF	189	EXP
151	POKE	190	COS
152	PRINT #	191	SIN
153	PRINT	192	TAN
154	CONT	193	ATN
156	LIST	194	PEEK
157	CLR	195	LEN
158	CMD	196	STR$
159	SYS	197	VAL
160	OPEN	198	ASC
161	CLOSE	199	CHR$
162	GET	200	LEFT$
163	NEW	201	RIGHT$
164	TAB 〈	202	MID$
165	TO	203	- 254 unused
166	SPC 〈	255	π

Der Aufbau einer BASIC Zeile ist wiefolgt:



Jede Zeile hat einen zwei Byte-Zeilennummer-Zeiger und einen zwei Byte langen Verbindungszeiger auf die nächste Zeile. Jede Zeile wird mit einer Null abgeschlossen. Innerhalb einer BASIC-Zeile kommt eine Null kaum vor, daher lassen sich BASIC-Zeilen verhältnismäßig einfach mit einem kleinen Programm auslesen und identifizieren.

```
63000 FORX=2048 TO 2070
63010 PRINT PEEK(X);
63020 NEXT X
```

Dieses Programm muß immer die gewünschte Endadresse und Anfangsadresse im FOR-Statement enthalten. Am besten Sie laden ein BASIC-Programm und sehen sich die ersten Zeile ab 2048 einmal an.

Als erstes sehen wir in 2048 die Null. Sie ist die Anfangsmarkierung. Dann folgt das lower Byte des Pointers zum nächsten Statement. In der nächsten Zelle ist das higher Byte des nächsten Statements.

Beispiel: ↓Zeiger auf Zelle 15

(0) (15) (4) (10) (0) (143) (32) (68) (65) (84)
(65) (32) (11) (1) (0) (29) (4)
↑ Zelle 15

Dieser Zeiger zeigt auf den Zeiger des nachfolgenden Befehls. In unserem Falle ist die Zelle 15 ab 0800 Hex (siehe Beispiel).

Programmbeispiel:

```
 10 REM DATA
100 POKE 59448,12
101 PRINT "♥QQQQ ....
```

Dann folgen lower und higher Byte der an dieser Stelle liegenden

Zeilennummer.

Will man die nächste Speicherzelle mit der Linkadresse finden zählt man in unserem Falle ab 15 einfach 15 Zellen weiter. Dort finden wir die Zahl 29. Von hier aus 29 Zellen weitergezählt finden wir nächstes "LINK"-Element, welches wiederum auf die nächste Zeile zeigt.

Die einzelnen Zahlen sind im BASIC Befehls-Teil als zwei Byte Werte abgelegt. D. h. POKE 53,3 würde im Befehlsteil wiefolgt aussehen:

151	/	53	/	51	/	44	/	51	/	0
↑		↑		↑		↑		↑		↑
POKE		5		3		Komma		3		Trennungsnull

Im Speicher ist natürlich vor dem Befehlsteil noch die Trennungs-Null, der LINK-Teil und die Zeilennummer abgelegt und am Ende folgt dann wieder eine Trennungsnull.

Wer sich diese Zusammenhänge gut klarmacht, versteht RENUMBER-Programme und Trace Routinen sowie Suchroutinen mit Referenzangaben besser. Sie können sich diese jetzt sogar vielleicht selbst schreiben. Auch ein Schutz gegen unberechtigtes Listen ist auf diese Weise möglich, indem man die Zeiger so verstellt, daß sie über REM-Statements miteinander gekoppelt werden und der dazwischenliegende Teil unsichtbar wird.

Das nachfolgende kleine Programme druckt Ihnen das jeweilige LINK-Feld und die zugehörige Zeilenzahl eines BASIC-Programmes aus.

```
10 A=2049
20 L=PEEK(A)+256*PEEK(A+1):IF L=0THEN END
30 PRINT"LINK FELD IST"L;
40 PRINT "ZEILENZAHL IST " PEEK(A+2)+256*PEEK(A+3)
50 A=L:GOTO 20
```

Eine andere kleine nützliche Routine druckt Ihnen diejenigen Zeilenzahlen aus, die den in Zeile Null aufgeführten Ausdruck enthalten. Z. B. in Zeile Null K = 1 eingegeben (0 K = 1) liefert Ihnen die Zeilenzahlen

0 und 62500 wenn Sie das nachfolgende Programm selbst nach Ausdrucken absuchen wollen.

```
62000 A=2049:B=256:J=2053:X=PEEK(J)
62010 P=PEEK(J):IF P=X THEN GOSUB 62500
62020 IF P<>0 THEN J=J+1:GOTO62010
62030 IF PEEK(J+2)=0 THEN END
62040 J=J+4:A=PEEK(A)+B*PEEK(A+1):GOTO62010
62500 K=1
62510 Y=PEEK(2053+K):IFY=0THEN PRINT PEEK(A+2)+
      B*PEEK(A+3):RETURN
62520 IF Y=PEEK(J+K)THEN K=K+1:GOTO 62510
62530 RETURN
```

Ein weiteres recht interessantes Programm dient dazu um Zeilen im Programm unsichtbar zu machen. Man kann damit wichtige Zeilen ausblenden und vor falschem Zugriff schützen.

```
50000 A=2049:B=256
50010 INPUT "VERSTECKE DIE ZEILE NACH";X
50020 FOR R=1TO 1E8:IF PEEK(A+2)+B*PEEK(A+3)<XTHEN A=PEEK(A)
      +B*PEEK(A+1):NEXT
50025 IF PEEK(A+2)+B*PEEK(A+3)>X THEN PRINT"KEINE ZEILE":END
50030 XS=A:REMANFANG VON ZEILE X
50040 YS=PEEK(A)+B*PEEK(A+1):REM ANFANG DER FOLGENDEN ZEILE
50050 X1=PEEK(YS):X2=PEEK(YS+1)
50060 POKE XS,X1:POKEXS+1,X2
50070 POKE YS+2,0:POKE YS+3,0
```

Geben Sie zuerst dieses Programm in Ihren Commodore 64 ein und probieren Sie dann die Funktion mit dem folgenden Beispiel aus.

```
10 REM BEISPIEL
20 PRINT"DIESES PROGRAMM"
30 PRINT"SOLL TEILWEISE"
40 PRINT"VERSTECKT WERDEN"
50 INPUT"NENNE EINE ZAHL";W
60 INPUT"NENNE EINE ZWEITE ZAHL";Z
70 R=W+Z:PRINT"SUMME";R
80 END
```

Starten Sie zuerst das Beispielprogramm und testen Sie es.

Starten Sie dann das Versteck-Programm mit RUN 50000 und geben Sie auf die Frage "Verstecke die Zeile nach" den Betrag 60 ein.

```
10 REM BEISPIEL
20 PRINT"DIESES PROGRAMM"
30 PRINT"SOLL TEILWEISF"
40 PRINT"VERSTECKT WERDEN"
50 INPUT"NENNE EINE ZAHL";W
60 INPUT"NENNE EINE ZWEITE ZAHL";Z
80 END
```

Beispielprogramm 2 mit versteckter Zeile 70

In Ihrem Beispielprogramm wird dann die Zeile 70 unsichtbar gemacht (siehe Beispielprogramm). Sie können dies noch weitertreiben und jede zweite Zeile verstecken. Das Programm steht dann wiefolgt im Bildschirm:

```
10 REM BEISPIEL
30 PRINT"SOLL TEILWEISE"
50 INPUT"NENNE EINE ZAHL";W
80 END
```

und liefert aber nach dem Start mit RUN noch die gleichen Ergebnisse wie unsere beiden ersten Programme. Auf diese Weise kann man seine eigenen Programme schützen.

Änderung der BASIC-Zeiger

Die BASIC-Zeiger beim Commodore 64 in den Speicherzellen 43 und 44 dezimal können natürlich abgeändert werden. Auf diese Weise kann der Anfang der BASIC-Programme an einen anderen Platz im Speicher gebracht werden. Man könnte z. B. ein Programm in den oberen RAM-Bereich ab C000 Hex = 49152 dez. ablegen. Hierzu müsste lower und higher Byte in die Adressen 43 und 44 gebracht werden.

```
POKE 43,0
POKE 44,192
POKE 49152,0
NEW
```

Diese Befehle lassen BASIC an der Adresse 49152 dez (C000 Hex) beginnen. Das Nullbyte am Anfang kennen wir von unseren Betrachtungen am Anfang her.

Eine Rückkehr zum "normalen" BASIC Anfang erreichen wir mit

POKE 43,0:POKE 44,8:POKE 2048,0:NEW

Mit NEW und CRL kann man die Pointer auf einfachste Weise festlegen. Neben den Anfangsadressen sollten auch die Zeiger für Variable, Strings und das BASIC Ende entsprechend überprüft werden.

MEMORY MAP

002B–002C	43–44	Zeiger: Beginn von BASIC
002D–002E	45–46	Zeiger: Beginn der Variablen
002F–0030	47–48	Zeiger: Beginn der Felder (Arrays)
0031–0032	49–50	Zeiger: Ende der Felder
0033–0034	51–52	Zeiger: Speicherung der Zeichenkette (nach unten)
0037–0038	55–56	Zeiger: Ende BASIC RAM

NOTIZEN

Dateien auf Cassette oder Diskette

Allgemeines

Im Handbuch (mitgeliefertes Handbuch) ab Seite 109 findet man ca. eineinhalb Seiten über Kassettenoperation. Dies sind die wichtigsten Grundlagen, die man braucht. Wenn man jedoch selbst ein Programm schreiben will, welches Daten auf einen externen Massenspeicher ablegen und auch wieder einlesen soll, benötigt man weit mehr Informationen.

Daten auf Cassette in BASIC

Die folgenden Ausführungen gelten auch für Diskettenbetrieb!
Der Commodore 64 erlaubt von BASIC aus zwei Arten der Informationsabspeicherung auf Cassette.

1. BASIC Programme mit LOAD und SAVE (PRG-Files)
2. Datenfiles von BASIC her (SEQ-Files)

BASIC-Programme werden in geschlüsselter Form ab Adresse 0800 Hex im Speicher abgelegt. Der Vorspann enthält Informationen über seine Adresse. Und so kann es auf Cassette gespeichert und anschließend wieder direkt in den Speicher geladen werden (an die Stelle wo es vorher gelegen war).

Daten sind ASCII-Zeichen, welche hintereinander aus einem PRINT-Befehl als String auf Cassette abgelegt werden. Sie können auch wieder zurück in den Speicher geladen werden. Aber nicht direkt, da der Rechner ja nicht weiß an welche Stelle sie geladen werden müssen. Aus diesem Grunde müssen Datenfiles beim Speichern und Laden über einen

Puffer geführt werden. Während der Puffer mit Daten beschrieben wird, werden gleichzeitig Daten auf die Cassette geschrieben. Ähnlich verhält es sich beim Einladen der Daten in den Rechner. Wir können dies später beobachten wie der Cassettenrecorder zum Einlesen bestimmter Datenblöcke immer ein- und dann wieder ausgeschaltet wird.

Speichern und Laden von Datenfiles

Wie zu Beginn schon erwähnt, ist die Behandlung von Datein bei Cassetten- und Diskettenbetrieb annähernd gleich.

Folgende Befehle stehen hierfür zur Verfügung:

Cassette	**Disk**
OPEN 1,1,1,"NAME"	OPEN 15,8,15,"NAME"
PRINT#1, A$	PRINT#15, A$
INPUT#1, A$	INPUT#15, A$
GET#1, X$	GET#15, X$
CLOSE 1	CLOSE 15
CMD	CMD

Es ist nicht ganz einfach mit Cassetten- oder Diskettenfiles zu arbeiten, wenn man nicht ein wenig damit geübt hat. Wir wollen deshalb im folgenden Teil einige Beispiele durchgehen. Wichtig dabei ist, daß wir die Beispiele so wählen, wie sie auch in der Praxis vorkommen können. Vorerst jedoch noch einige grundlegende Angaben.

Daten können wiefolgt auf Cassette gespeichert werden:

```
 5 INPUT A$
10 OPEN 1,1,1"NAME"
20 PRINT#1, A$
30 CLOSE 1
```

Daten können wiefolgt von Cassette gelesen werden:

```
40 OPEN 1
50 INPUT#1, A$
60 CLOSE 1
70 PRINT A$
```

Mit diesen beiden kleinen Programmen kann ein String A$ unter dem Namen A$ abgelegt werden und unter A$ wieder von Cassette gelesen werden. Nach erfolgreichem Ladevorgang wird der String A$ ausgedruckt.

Dies sieht ja alles ganz einfach aus. Aber in der Praxis hat man viele Daten und mit einem String kommt man ja wohl selten aus. Obwohl diese beiden Beispielprogramme schon arbeiten, treten jetzt viele weitere Fragen auf.

1. Wie lang darf mein String überhaupt sein ?
2. Wieviele Strings kann ich überhaupt eingeben ?
3. Wo ist der String, bevor ich es auf Cassette lade ?
4. Wenn ich mehrere Strings habe, wie stelle ich fest wo welcher String ist ?
5. Was geschieht mit Zahlen ?
6. Was ist der Unterschied zwischen INPUT# und GET# ?
7. Wie erkenne ich wo Text aufhört und ab wann ich nicht mehr laden oder lesen muß ?

Sie sehen die Fragen häufen sich. Schlimmer wird die Sache noch, wenn man selbst Programme entwickelt und Informationen sucht. Bald stellt man fest, daß gerade die Information die man jetzt selbst braucht, nirgends zu finden ist. Aus diesem Grunde wollen wir hier die Techniken so ausführlich wie möglich besprechen.

Zuerst zu den Fragen. Wie lange ein String sein darf stellen Sie bei Ihrem Commodore 64 wiefolgt fest:

```
10 INPUT "STRING" A$
20 PRINT A$
```

Geben Sie nach RUN die Zahlen 1234567890 zwei Zeilen lang ein und drücken Sie RETURN. Sie sehen ein String kann bei der Eingabe über die Tastatur bis zu 80 Zeichen lang sein. Dies kommt daher, daß der BASIC Eingabepuffer nur 88 Zeichen groß ist und nur 2 Zeilen a' 40 Zeichen eingegeben werden können. Alles was darüber hinaus an Zeichen eingegeben wird bewirkt, daß die ersten 80 Zeichen abgeschnitten werden und nur noch die Zeichen in der Stringvariablen

stehen, die nach dem 80. Zeichen eingegeben wurden.

Innerhalb des Rechners können Strings bis zu 226 Zeichen lang sein. Im Commodore 64 BASIC müssen Strings nicht gesondert dimensioniert werden. Das System übernimmt die Erkennung. Im Gegensatz zu anderen BASIC-Versionen bedeutet beim Commodore 64 BASIC: DIM A$(20) nicht daß A$ ein String mit maximal 20 Zeichen Länge ist, sondern A$(20) ist ein Stringarray mit 21 Stringelementen von A$(0) bis A$(20).

Die Daten im Rechner müssen also vor dem Abspeichern in Stringvariablen gespeichert sein. Von dort aus, nach dem Öffnen eines Files werden zuerst Daten aus dem Bandpuffer (033AC — 03FB Hex (828 — 1019 Dez) mit dem Vorspann und einigen Identifikationsadressen auf Band geschrieben. Dann werden die Stringvariablen über den Bandpuffer auf Band geschrieben. Da der Bandpuffer nur 191 Zeichen groß ist, sollten die einzelnen Strings nicht länger als 32 Zeichen sein. Hat man größerer Strings so empfiehlt es sich diese in kleinere Einheiten von je 32 Zeichen Länge zu zerlegen und später wieder zusammenzufügen. Es hat sich daher als praktisch erwiesen, wenn man die Daten vorher in ein Stringarray packt und dann die einzelnen Elemente aus dem Stringarray heraus auf Band liest.

Hier ein kleines Beispiel zur Erläuterung:

Mit dem ersten kleinen Programm können Sie ein Array mit 33 Elementen mit Namen füllen und diese dann auf dem Bildschirm ausdrucken. Die Eingabe wird beendet, indem wir das £ Zeichen anstelle eines weiteren Namens eingeben.

Bitte beachten Sie, daß nach Abbruch des Programmes und Neustart mit RUN alle Stringvariablen gelöscht werden. Mit GOTO 〈 Zeilennummer 〉 kann man im Notfall die Variablen retten.

```
10 REM CASSETTEN FILE DEMO
15 DIM A$(32)
17 FOR I=0 TO 31
20 INPUT"NAME";A$(I)
30 IF A$(I)="£" THEN 50
40 NEXT I
50 PRINT"ENDE DER EINGABE"
```

```
55 FOR B=0TO 2000:NEXT B
60 REM AUSGABE AUF BILDSCHIRM
70 FOR I=0 TO 32
80 PRINT" ";A$(I);
90 NEXT I
100 END
```

Bei der Ausgabe auf den Bildschirm haben wir uns keine Gedanken um das Ende der Daten gemacht. Man könnte jedoch jetzt auch hier nach dem £ Zeichen abfragen und die Schleife verlassen. Bis zu 31 Eingaben plus das Endzeichen £ können also mit diesem Programm in unserem Computer gespeichert werden. Wollten wir die eingegebenen Namen auf Cassette speichern, müssten wir unser Programm wiefolgt ändern.

```
10 REM CASSETTEN FILE DEMO
15 DIM A$(32)
17 FOR I=0 TO 31
20 INPUT"NAME";A$(I)
30 IF A$(I)="£" THEN 50
40 NEXT I
50 PRINT"ENDE DER EINGABE"
55 FOR B=0TO 2000:NEXT B
60 REM AUSGABE AUF CASSETTE
65 INPUT"WOLLEN SIE AUF CASSETTE";Z$
67 IF Z$="JA" THEN 120
70 FOR I=0 TO 32
80 PRINT A$(I)
85 IF A$(I)="£" THEN 100
90 NEXT I
100 END
120 INPUT"FILENAME";Y$
130 OPEN 1,1,2,Y$
140 I=0
160 PRINT#1,A$(I)
165 IF A$(I)="£" THEN 180
167 I=I+1
170 GOTO 160
180 CLOSE 1
190 END
```

Das abspeichern (SAVE) auf Cassette ist relativ problemlos. Die Daten

müssen in Form von Variablen (Zahlen) oder Stringvariablen aus einem Array (Feld) vorliegen. Nach öffnen des Cassettenfiles mit OPEN 1,1,2,Y$ werden die einzelnen Elemente mit PRINT# 1, A$(I) auf Band geladen (einschließlich der CR-RETURN). Bei der Eingabe ins Array (siehe ab Zeile 20) haben wir als letztes Zeichen zur Begrenzung der Eingabe ein £ Zeichen eingegeben. Dieses dient uns jetzt beim Schreiben auf Band auch wieder als Endmarkierung (End of File-Marker). Werden mehr als 40 Zeichen ohne CR-RETURN ausgegeben, so wird am Zeilenende auch ein CR-RETURN angefügt. Jedes File sollte mit einem Namen versehen werden.

Wichtige Regeln beim Schreiben auf Band

1. Beachten Sie, daß die INPUT# und PRINT# Variablenlisten gleich sind. Wenn die INPUT Liste zu kurz ist, verlieren Sie Daten.
2. Geben Sie über PRINT# niemals mehr als 79 Zeichen ohne ein CR-RETURN aus.Wenn Sie später mit INPUT# ein Zeichen mit mehr als 80 Zeichen lesen wollen, erhalten Sie einen STRING TOO LONG ERROR. Frühere BASIC-Versionen stürzten ab und der Rechner musste neu eingeschaltet werden.
3. Zusätzliche CR-RETURNs schaden nie !
4. Wenn Sie mehrere Zahlen in eine Zeile schreiben wollen, trennen Sie diese durch ein Komma ",". Anderenfalls erhalten Sie beim Einlesen falsche Zahlenwerte.

Lesen von Cassette (LOAD)

Das Einlesen vom Band ist wesentlich komplizierter. Daten von Cassette oder Diskette können mit den Befehlen INPUT# oder GET# eingelesen werden.

Der Befehl INPUT# [logische File Nr.], [variable List]

INPUT#
erkennt ASCII-Zeichen mit CR (Carriage RETURN) als Trennelement. Es erkennt nicht die Zeichen des Bildschirmeditors, es sei denn, es geht ein Anführungsstrich CHR$(34) voraus, INPUT# kann nur Strings kleiner als 80 Zeichen vom Band lesen. Dies hängt mit der größe des BASIC-Eingabepuffers 512 – 600 Dez beim Commodore 64 zusammen.

Werden Werte größer 89 Zeichen über INPUT# hereingelesen, erfolgt eine Unterbrechung des Programmes mit "STRING TOO LONG ERROR". Mit CR-RETURN CHR$(13) oder $0D kann ein Record auf ähnliche Weise wie bei der Eingabe über Tastatur mit RETURN abgeschlossen wird, abgetrennt werden. In ähnlicher Weise werden Kommas und Punkte, wenn nicht ein Anführungszeichen vorangeht, behandelt. Das Programm erkennt einen Punkt ohne vorangehendes Anführungszeichen als Trennelement ähnlich dem CR-RETURN.

Dies führt dann zu Problemen, wenn bei der Eingabe solche Zeichen als Trennelemente aufgefasst werden und die zugewiesenen Variablen nicht ausreichen, um die einzelnen Elemente aufzunehemen.

INPUT# liest die Daten vom Band, als wäre der Cassettenrecorder die Tastatur. Auf diese Weise können auch nicht mehr als 79 Zeichen plus dem CR-RETURN hereingelesen werden. Dies bedeutet, daß auf dem Band niemals mehr als 79 aufeinanderfolgende Zeichen ohne CR-RETURN mit INPUT# eingelesen werden können.

Der Befehl GET# [logische File Nr.], [String oder numerische Variable]

GET#
liest immer nur ein Zeichen nach dem anderen vom Band oder von der Disk. Stringvariable oder numerische Variable können damit einzeln eingelesen werden. Wenn Files mit GET# später eingelesen werden sollen, braucht man beim Schreiben auf Band nach spätestens 79 Zeichen keine CR-RETURNS einzufügen.

Im folgenden Beispielprogramm lesen wir mit dem GET# Befehl die Daten nacheinander vom Band. Wir lesen jedes Zeichen einzeln mit C$ und packen es in einen String T$. Wenn dieser String 32 Zeichen lang ist, schließen wir diesen ab und übernehmen ihn in ein Feld A$(J). Unsere Daten liegen dann in einem Feld mit 32 Elementen (siehe Zeile 15) im File Demo Programm.

Wir könnten die einzelnen Feldelemente auch größer als 32 machen, wenn wir dazu die Abfrage in Zeile 260 ändern. Auch die Anzahl der Felder könnte durch eine Änderung der Dimensionierung in Zeile 15 abgewandelt werden.

```
200 REM EINLESEN EINES CASS.FILES
210 OPEN 1
220 J=0
222 I=0:T$=""
250 GET#1,C$:IF C$="£" THEN 275
255 T$=T$+C$:I=I+1
260 IF I=32 THEN A$(J)=T$:J=J+1:GOTO222
270 GOTO 250
275 T$=T$+C$:I=I+1
277 A$(J)=T$
280 CLOSE 1
285 I=0:J=0
295 FOR I=0 TO 32
300 PRINT A$(I);
305 IF A$(I)=""THEN GOTO 320
310 NEXT I
320 END
400 OPEN 1
410 I=0
```

Während des Einlesens prüfen wir jedes Zeichen, ob es ein £ Zeichen ist. Dies ist unsere Endmarkierung (siehe Zeile 250). Wenn das £ erreicht ist, schreiben wir das letzte Element noch voll und legen es im Array (Feld) ab. Dann wird das File geschlossen und die gelesenen Werte zur Kontrolle auf dem Bildschirm ausgedruckt.

Nachfolgend sehen Sie noch einmal das gesamte Programm, welches als Ausgangsbasis für Ihre eigenen Experimente gedacht ist. Es arbeitet einwandfrei und erlaubt mit RUN die Eingabe von Namen oder Daten, die bis zu 79 Zeichen lang sein können. Die Eingabe wird durch £ und RETURN abgebrochen. Sie werden dann gefragt, ob Sie auf Cassette ablegen wollen. Wenn Ja, antworten Sie mit JA. Alle eingegebenen Daten werden dann auf Cassette gespeichert. Als Filename kann eine fünfstelliger Name eingegeben werden.

Komplettes Demo Programm

```
10 REM CASSETTEN FILE DEMO
15 DIM A$(32)
17 FOR I=0 TO 31
20 INPUT"NAME";A$(I)
30 IF A$(I)="£" THEN 50
40 NEXT I
50 PRINT"ENDE DER EINGABE"
55 FOR B=0TO 2000:NEXT B
60 REM AUSGABE AUF CASSETTE
65 INPUT"WOLLEN SIE AUF CASSETTE";Z$
67 IF Z$="JA" THEN 120
70 FOR I=0 TO 32
80 PRINT A$(I)
85 IF A$(I)="£" THEN 100
90 NEXT I
100 END
120 INPUT"FILENAME";Y$
130 OPEN 1,1,2,Y$
140 I=0
160 PRINT#1,A$(I)
165 IF A$(I)="£" THEN 180
167 I=I+1
170 GOTO 160
180 CLOSE 1
190 END
200 REM EINLESEN EINES CASS.FILES
210 OPEN 1
220 J=0
222 I=0:T$=""
250 GET#1,C$:IF C$="£" THEN 275
255 T$=T$+C$:I=I+1
260 IF I=32 THEN A$(J)=T$:J=J+1:GOTO222
270 GOTO 250
275 T$=T$+C$:I=I+1
277 A$(J)=T$
280 CLOSE 1
285 I=0:J=0
295 FOR I=0 TO 32
300 PRINT A$(I);
305 IF A$(I)=""THEN GOTO 320
310 NEXT I
320 END
```

Der Commodore 64 und die Diskettenstation 1541

1. Allgemeines

Nachdem ich seit 1978 (also mehr als 5 Jahre) ausschließlich mit der DATASETTE gearbeitet habe, können Sie sich vorstellen, wie froh ich war eindlich mit einer Diskettenstation für ein Commodore System arbeiten zu können.

Die 1541 Disk Station wird komplett mit eingebautem 220V Netzteil, Kabel zum Anschluß an den RS232 Port des Commodore und einer Demo Diskette mit DOS 5.1 geliefert.

Die Station erlaubt auf einer Diskette ca. 170k Byte zu speichern. Der serielle Datenverkehr macht das Arbeiten mit Disk nicht besonders schnell, aber für den allgemeinen Bedarf reicht es völlig aus. Ein Programm mit 5000 Byte ließ sich in 15 Sekunden von Diskette in den Speicher laden. Die Diskettenstation enthält einen eigenen 6502 Prozessor sowie zwei 6522 VIA-Bausteine. Weiterhin befindet sich ein eigener Pufferspeicher (2k 2114) in der Diskettenstation.

Der Disk-Controller sowie die zugehörige Software (16k ROM) machen das System zu einer intelligenten Diskettenstation. Aus diesem Grunde wird durch den Anschluß der 1541 an den Commodore 64 kein Speicherplatz im Commodore 64 beansprucht. Während Sie eine Diskette formatieren können Sie gleichzeitig ein Programm von Cassette einlesen.

Ich kann jdem, der nur im entferntesten irgend welche Geschäftssoftware verwenden will zu einer 1541 raten. Kein Geschäftsmann kann sich den Ärger mit Cassetten und den Zeitaufwand heute noch leisten.

2. Anschluß des Systems

Wie das System angeschlossen wird, ist im Handbuch ganz gut beschrieben. Wichtig ist, daß der Commodore 64 immer als letztes eingeschaltet wird. Z. B. zuerst Drucker, dann Disk und dann den Commodore 64.

Die 1541 Disk-Station lässt sich auch an den VC-20 anschließen. Die

Geschwindigkeit der eingehenden Daten sind hier etwas unterschiedlich. Eine Anpassung kann Softwaremäßig mit

OPEN 15,8,15,"UI–":CLOSE 15

durchgeführt werden.

Meinen VC-1515 Drucker habe ich auf Device 4 gestellt und mit
OPEN 4,4
CMD 4
im Direktmodus ansprechen können. Wenn Sie nach der Druckerinitialisierung wieder einen Diskettenzugriff machen wollen, müssen Sie zuerst das File wieder mit CLOSE 4,4 wieder schließen. Ansonsten gibt es eine Fehlermeldung.

3. Erste Kontakte

Das mitgelieferte englische Handbuch ist für den der bereits einmal mit Commodore Diskettenstationen gearbeitet hat, sicher ausreichend. Der Anfänger tut sich sicher etwas schwer. Ich möchte deshalb hier die wichtigsten Punkte zusammenfassen und in der Reihenfolge besprechen, wie man meist mit einem solchen System zu arbeiten beginnt.

Wenn alles ordnungsgemäß angeschlossen und eingeschaltet ist, kann die Diskette, mit dem Beschriftungsschild nach oben, in die Diskettenstation eingelegt werden.
Der Commodore 64 wird eingeschaltet und es erscheint die gewohnte BASIC Ausgangsinformation am oberen Bildschirmende.
Da kein DOS geladen werden muß geht das alles recht schnell. Man kann jetzt wie gewohnt mit LOAD und SAVE mit der Datasette arbeiten oder auch als ersten Diskettenzugriff mit:

LOAD "$",8

das Diskettendirectory von Diskette in den Speicher holen. Mit LIST erscheint dies dann auf dem Bildschirm

OPEN 4,4:CMD 4:LIST

brachte mir dann das Directory auf den Drucker.

```
0 [illegible]
13   "HOW TO USE"          PRG
5    "HOW PART TWO"        PRG
4    "VIC-20 WEDGE"        PRG
1    "C-64 WEDGE"          PRG
4    "DOS 5.1"             PRG
11   "COPY/ALL"            PRG
9    "PRINTER TEST"        PRG
4    "DISK ADDR CHANGE"    PRG
4    "DIR"                 PRG
6    "VIEW BAM"            PRG
4    "CHECK DISK"          PRG
14   "DISPLAY T&S"         PRG
9    "PERFORMANCE TEST"    PRG
5    "SEQUENTIAL FILE"     PRG
13   "RANDOM FIAL"         PRG
558 BLOCKS FREE.
```

Die Diskette bietet Platz für 664 Blocks. Jeder Block hat 256 Byte. Links neben dem Namenseintrag im Directory finden Sie Anzahl der belegten Blöcke.

Nach der Ausgabe auf dem Drucker das File wieder mit CLOSE 4,4 schließen!

Die mitgelieferte Diskette enthält eine einfache Einführung, wie man die einzelnen Programme auf der Diskette verwendet, sowie die VC-20 und Commodore 64 Wedge Programme. Dies sind eine Art Aufrufprogramme für das DOS Unterstützungsprogramm DOS 5.1, welches sich auch auf der Diskette befindet und auch relativ wenig Platz beansprucht.

Auf die anderen Befehle will ich später noch kurz eingehen. Der erste Gedanke, nachdem man die ersten Versuche mit der Diskettenstation gemacht hat, ist, wie fertige ich eine Kopie meiner Masterdiskette an. Wer nur eine Diskettenstation hat, kann das Utility Programm "COPY/ALL" nicht verwenden. BASIC Programme lassen sich mit LOAD und SAVE auf eine andere Diskette bringen. Bevor Sie jedoch etwas auf eine neue Arbeitsdiskette abspeichern können, müssen Sie diese zuerst formatieren.

Dies geschieht mit dem folgenden Befehl:

OPEN 15,8,15 im Direktmode
PRINT #15, "NEW0:NAME, ID"

0 = Disk 1 (erste Diskettenstation)
Name = beliebiger Name
ID = Ein Identifikations Kurzzeichen bestehend aus zwei Zeichen

Der Name und die Identifikation erscheint später immer als Überschrift im Directory.

Das Formatieren einer Diskette dauert ungeführ 1 Minute. Meine Disk gibt zu Beginn des Formatierungsvorganges immer verdächtige Geräusche ab. Diese scheinen jedoch keinerlei Auswirkungen zu haben.

Wenn die Diskette initialisiert ist, können Sie bereits Programme darauf abspeichern. Es empfiehlt sich jedoch vorher mit LOAD und SAVE die Files "C-64 WEDGE" und "DIR" auf die neue Diskette zu laden. Diese brauchen Sie in jedem Falle, um ein Directory Ihrer Diskette auf den Bildschirm oder Drucker zu bringen. Das Kopieren dieser BASIC Files ist einfach und geschieht wiefolgt:

	LOAD"C–64 WEDGE",8
Diskettenwechsel	SAVE"C–64 WEDGE",8

Man braucht für den SAVE-Vorgang die Zeile nicht unbedingt wieder einzugeben. Gehen Sie mit dem Cursor an den Zeilenanfang der LOAD Zeile zurück und überschreiben Sie einfach LOAD mit SAVE und drücken Sie RETURN. In einigen Fällen hatte ich Probleme nach der Formatierung ein Programm auf die neue Disk zu schreiben. Das Programm stand dann zwar im Directory, ließ sich aber nicht mehr wieder laden. Die rote LED an der Disk begann zu blinken und signalisierte einen Fehler.

Wichtig ist, daß man nach der Initialisierung das File wieder mit CLOSE 15,8,15 schließt, bevor man irgend welche anderen Programm- oder Filezugriffe beginnt.

Ähnlich wie beim ATARI kann das Laden von Programmen von der

Diskette mit einem Asterik (*) etwas vereinfacht werden. Namen können abgekürzt werden.

Anstelle von	LOAD "NAME", 8	kann auch
	LOAD "NA*",8	eingegeben werden.

Das erste Programm bei dem jetzt die beiden Buchstaben NA übereinstimmen, wird von der Diskette in den Speicher geladen.

Weitere interessante Befehle und Funktionen

Die 1541 Diskettenstation hat weitere interessante Befehle, die das Arbeiten mit dem System sehr vereinfachen.

1. SAVE und REPLACE
2. VERIFY
3. COPY (Merge)
4. RENAME
5. SCRATCH
6. INITIALIZE
7. VALDATE
8. DUPLICATE (hier für zwei Disk-Drives)
9. SEQUENTIELLE FILEBEHANDLUNG
10. RANDOM ACCESS FILES (Files mit wahlfreien Zugriff)
11. OPEN, PRINT#, CLOSE

Files auf Diskette

Wie zu Anfang dieses Kapitels schon angedeutet, unterscheidet sich die Behandlung von Files auf Disketten nur unwesentlich von der Cassettenfilebehandlung.

Wie bei der Cassette werden auch mit den Befehlen PRINT#, INPUT# und GET# Daten auf die Diskette geschrieben und wieder zurückgelesen.

Versuchen wir einmal unser vorangegangenes Demoprogramm wiefolgt abzuändern.

```
130 OPEN 2,8,2,"0:"+Y$+",S,W"

READY.

160 PRINT#2,A$(I)

READY.

180 CLOSE 2

READY.

210 OPEN 2,8,2,"0:"+Y$+",S,R"

READY.

250 GET#2,C$:IF C$="£" THEN 275

READY.

205 INPUT"FILENAME";Y$

READY.
```

Dieses Programm schreibt genau wie das vorher gezeigte Cassettendemoprogramm ein Feld (Array) auf Diskette. Starten Sie einfach mit RUN, wenn Sie auf Diskette schreiben wollen. Starten Sie mit RUN 200, wenn Sie ein File von Disk lesen wollen. Auf die Frage "Filename" geben Sie einen Filenamen ein. Z. B. OLLI. Interessant ist, daß Sie später im Directory folgenden Eintrag finden:

```
1     "OLLIS"          SEQ
```

Disketten-File Demo

```
10 REM DISKETTEN FILE DEMO
15 DIM A$(32)
17 FOR I=0 TO 31
20 INPUT"NAME";A$(I)
30 IF A$(I)="£" THEN 50
40 NEXT I
50 PRINT"ENDE DER EINGABE"
55 FOR B=0TO 2000:NEXT B
60 REM AUSGABE AUF DISKETTE
65 INPUT"WOLLEN SIE AUF DISKETTE";Z$
67 IF Z$="JA" THEN 120
70 FOR I=0 TO 32
80 PRINT A$(I)
85 IF A$(I)="£" THEN 100
90 NEXT I
100 END
120 INPUT"FILENAME";Y$
130 OPEN 2,8,2,"0:"+Y$+",S,W"
140 I=0
160 PRINT#2,A$(I)
165 IF A$(I)="£" THEN 180
167 I=I+1
170 GOTO 160
180 CLOSE 2
190 END
200 REM EINLESEN
205 INPUT"FILENAME";Y$
210 OPEN 2,8,2,"0:"+Y$+",S,R"
220 J=0
222 I=0:T$=""
250 GET#2,C$:IF C$="£" THEN 275
255 T$=T$+C$:I=I+1
260 IF I=32 THEN A$(J)=T$:J=J+1:GOTO222
270 GOTO 250
275 T$=T$+C$:I=I+1
277 A$(J)=T$
280 CLOSE 2
285 I=0:J=0
295 FOR I=0 TO 32
300 PRINT A$(I);
```

```
305 IF A$(I)=""THEN GOTO 320
310 NEXT I
320 END
400 OPEN 1
410 I=0
```

Wer das £ Zeichen nicht auf dem Bildschirm mit ausgeben will, muß die Zeile 250 folgendermaßen abändern:

↓kein Leerzeichen dazwischen

```
250 GET#2,C$:IF C$="£" THEN C$="":GOTO 275
```

Wie auch bei den Cassettenfiles gibt es eine Save und Replace Anweisung. D. h. wir können das Array auch unter dem gleichen Namen mit neuen oder alten Inhalt wieder auf Diskette laden. In Zeile 130 wird hierfür einfach noch ein @ (Klammeraffe) eingefügt.

```
130 OPEN 2,8,2,"@0:"+Y$+",S,W"
```

Wenn man von Anfang an weiß, daß man ein Array öfter unter dem gleichen Namen auf Disk ablegen und lesen will, kann man von Anfang an die Zeile 130 so wie oben gezeigt festlegen. Auf diese Weise können Sie dann in das File mit dem Namen Y$ beliebig hinein schreiben und herauslesen.

Das Disketten-File Demo ist Ihnen jetzt eine gute Hilfe für Programmänderungen. Z. B. können Sie sich leicht unseren Wortprozessor WORT64, der normalerweise mit Cassettenfiles arbeitet, auf Diskettenbetrieb umändern. Die Schreib- und Leseroutinen sind ähnlich aufgebaut. Genauere Einzelheiten entnehmen Sie bitte dem VC 1541-Disketten User Manual.

Alle Files (Dateien) die wir bis jetzt besprochen haben waren serielle Files. Bei seriellen Files werden alle Daten hintereinander auf Diskette oder Cassette gespeichert und auch so wieder in den Speicher gelesen. Bei seriellen Files, wird das File beim Lesen immer vom Anfang bis zum Ende, oder bis zu einem durch Abfrage herbeigeführten Abbruch gelesen. Das File wird dann wieder mit CLOSE geschlossen. Auch der

Schreibvorgang erfolgt bei jedem WRITE (W) Zyklus am Anfang der Datei und überschreibt die gesamte vorhandene Datei neu.

Serielle Files eignen sich bei solchen Anwendungen ganz gut, bei denen große Datenmengen jedesmal auf einmal abgespeichert oder eingeladen werden sollen. Wenn man jedoch Programme schreiben will, die nur Teile einer großen Datenmenge ansprechen sollen, verwendet man besser Files mit wahlfreiem Zugriff. Files mit wahlfreiem Zugriff (Random Files) erlauben es beim C-64 gezielt auf Daten auf der Diskette zuzugreifen. Man braucht also nicht jedesmal das gesamte File von Anfang an her durchzusuchen, bis man die entsprechende Information findet oder das gesamte File von Anfang an her durchzusuchen, bis man die entsprechende Information findet oder das gesamte File neu schreiben, nur weil man am Ende einige wenige Daten anhängen will. Auf diese Weise spart man erheblich Zeit beim Zugriff auf Daten und schont den Schreib/Lesekopf der Diskettenstation.

Ton und Grafik mit dem Commodore 64

Allgemeines

Der Commodore 64 besitzt zwei sehr hochentwickelte Spezialbausteine zur Tonerzeugung und Grafikbehandlung.

1. Den 6567 Grafikbaustein
2. und den 6581 Ton-Controller-Baustein.

Die beiden integrierten Schaltungen bieten eine Reihe von Vorzügen, die wir in keinem anderen Personalcomputer finden. Ein großer Mangel seitens des Herstellers wird jedoch offenkundig, wenn man feststellt, daß es kaum Befehle in BASIC gibt, welche einen unkomplizierten Zugriff auf die hervorragenden Eigenschaften ermöglichen.

Man ist also auf sich selbst gestellt und muß über PEEK und POKE die gesamten Daten selbst in den Registern managen. Grund hierfür ist, daß man den C-64 zwar von der Hardwareseite recht interessant konzipiert hat, aber einfach ein BASIC aus vorherigen PET/CBM Maschinen einfach in den C-64 "genagelt" hat.

Durch die Flexibilität des C-64 ist dies nicht einmal schlimm, da man später auf einfachste Weise ein neues mit leistungsfähigen Grafik-PLOT und DRAW Befehlen und mit entsprechenden Kommandos für die Tonausgabe ausgestattetes BASIC leicht von Cassette oder Diskette nachladen kann. Natürlich ist es auch möglich dieses neue BASIC in ROM in den Erweiterungsport zu stecken oder gleich im Rechner durch Auswechseln der ROMs auszutauschen.

Der VIC-6567 Baustein im C-64 und seine Eigenschaften

Der VIC-6567 erlaubt dem Commodore 64 insgesamt 16 Farben darzustellen. Die maximale Auflösung beträgt 320 x 200 Bildpunkte pro Bildschirm. 8 Sprites, ähnlich den Player Missiles beim ATARI, können gleichzeitig auf den Bildschirm gebracht werden.

Jeder Sprite besteht aus einem Feld aus 24 x 21 Bildpunkten. Die Sprites können in unterschiedlichen Farben festgelegt werden. Sprites die mehrere Farben enthalten sind auch möglich. Es wäre jetzt überflüssig alle Zusammenhänge noch einmal zu beschreiben. Sie finden diese ausführlich und gut im "Commodore-64 Micro Computer Handbuch" ab Seite 58 erläutert.

Wir wollen deshalb an Hand kleiner praktischer Beispiele den Stoff nur kurz erläutern.

Kurze Zusammenstellung der wichtigsten Einzelheiten (Blitzkurs)

Um ein Zeichen auf dem Bildschirm in Farbe darzustellen, gibt es drei Möglichkeiten:

1. Über die Tastatur durch Betätigen der entsprechenden Taste

CTRL 3 ergibt rot
CTRL 4 ergibt türkis

1 PRINT "CTRL1E CTRL2L CTRL3C CTRL4O CTRL5M CTRL6P"

```
1 PRINT"■E■L■C■O■M■P"
```

Die CTRL Befehle erscheinen im Statement als grpahisches Zeichen, ähnlich wie wir es von CLEARHOME oder von den CURSOR Befehlen innerhalb Print Anweisungen her kennen. Es wird immer die momentane Cursorposition eingefärbt. Die Farbe bleibt, bis sie wieder geändert wird.

2. Eine übersichtlichere Darstellung ist die Form PRINT CHR$(x).

Hierzu benötigt man die Tabelle "ASCII und CHR$ Codes" auf Seite 135 im Commodore Microcomputer Handbuch. Unser obiges Beispiel

würde dann wiefolgt aussehen.

```
2 PRINTCHR$(5);"E";CHR$(28);"L";CHR$(30);"C";CHR$(159);"O";
3 PRINTCHR$(30);"M";CHR$(5);"P"
4 END
```

3. Farbe durch Poke

Da jeder Speicherplatz innerhalb des Bildschirmspeichers über den POKE-Befehl verändert werden kann, können wir auch beliebige Zeichen in den Bildschirmspeicher bringen.

```
10 POKE 1144,83
```

Es erscheint jeweils in der Farbe die gerade für die momentane Cursorposition gültig ist. Um dieses Zeichen auch in verschiedenen Farben darstellen zu können, hat der C-64 neben dem eigentlichen Bildschirmspeicher noch einen zweiten Farbspeicher.

Bildschirmspeicher:	1024 – 2023 Dez	Hex 0400 – 07E7
Farbspeicher:	55296 – 56295 Dez	Hex D800 – DBE7

Wenn wir unser Zeichen oben farbig darstellen wollen, müssen wir noch

```
10 POKE 1144,83
20 POKE 55416,2:REM ROT
```

eingeben. Die Hintergrundfarbe wird durch POKEN in ein 6567 Register dargestellt.

POKE 53281,x = Hintergrundfarbe
POKE 53280,x = Farbe des Rahmens

Die Farbenwerte können Sie folgender Tabelle entnehmen. Sie gelten für alle POKE Befehle in den Farbspeicher des C-64.

POKE Werte	CHR$(x) Werte
0 = schwarz	144
1 = weiß	5
2 = rot	28
3 = türkis	159
4 = violett	156
5 = grün	30
6 = blau	31
7 = gelb	158
8 = orange	
9 = braun	
10 = hellrot	
11 = grau 1	
12 = grau 2	
13 = hellgrün	
14 = hellblau	
15 = grau 3	

Unser vorhergehendes Beispiel würde mit POKE-Befehlen aufgebaut wiefolgt aussehen:

```
30 POKE 53272,23
40 POKE 1144,69:POKE 55416,1
50 POKE 1146,67:POKE 55418,5
60 POKE 1147,79:POKE 55419,3
80 POKE 1148,77:POKE 55420,5
90 POKE 1149,80:POKE 55421,1
```

Zum Starten dieses Programmes löschen Sie den Bildschirm und bewegen den Cursor an den unteren Bildschirmrand.

Wir haben uns jetzt ausschließlich mit der Erzeugung von farbigen Buchstaben oder Grafikzeichen aus dem Zeichensatz des C-64 beschäftigt. Sie können sich diese auf dem Bildschirm mit

FOR I = 0 TO 255 : PRINT CHR$(I) : NEXT I

												+10										+20										+30									
Adresse		0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
1024																																									
1064																																									
1104																																									
1144																																									
1184																																									
1224																																									
1264																																									
1304																																									
1344																																									
1384																																									
1424																																									
1446																																									
1504																																									
1544																																									
1584																																									
1624																																									
1664																																									
1704																																									
1744																																									
1784																																									
1824																																									
1864																																									
1904																																									
1944																																									
1984																																									

BILDSCHIRMRAM

												+10										+20										+30									
Adresse		0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
55296																																									
55336																																									
55376																																									
55416																																									
55456																																									
55496																																									
55536																																									
55576																																									
55616																																									
55656																																									
55696																																									
55736																																									
55776																																									
55816																																									
55856																																									
55896																																									
55936																																									
55976																																									
56016																																									
56056																																									
56096																																									
56136																																									
56176																																									
56216																																									
56256																																									

FARBRAM

ausgeben und ansehen.

Neben einer Grafik mit dem Zeichen aus dem Zeichensatz lässt sich der C-64 auch in hochauflösender Grafik programmieren.

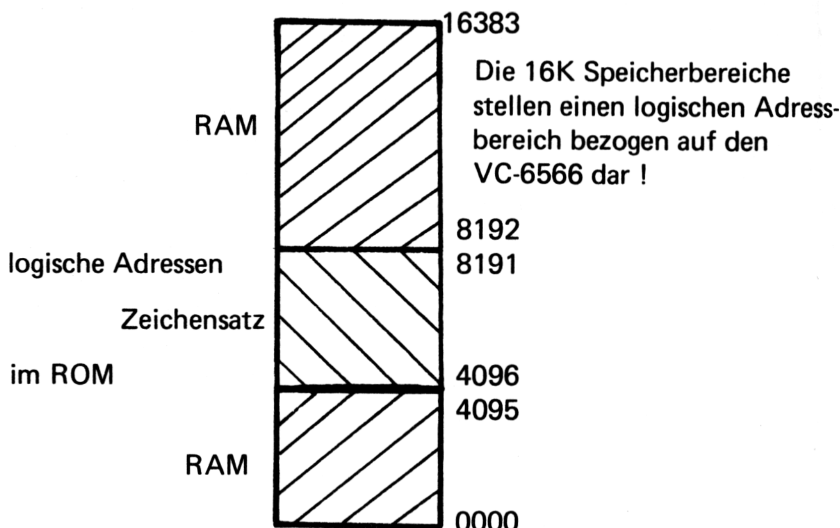
Um diese Zusammenhänge besser verstehen zu können, wenden wir uns zuerst dem eingebauten Zeichensatz und dem Aufbau der einzelnen Bildschirmzeichen zu. Der C-64 kann im Zeichensatzmodus 25 Zeilen a' 40 Zeichen auf dem Bildschirm gleichzeitig darstellen. Dieser Zeichensatzmodes stellt sich nach dem Einschalten automatisch ein. Der benötigte Bildschirmspeicher beträgt 1024 Bytes. Alle Funktionen die zur Kontrolle dieser Zeichen- und Bildschirmbehandlung notwendig sind übernimmt der 6560 VIC-Chip. Kontrollfunktionen für diesen Chip finden Sie in den Registern der beiden 6526 I/O-Bausteine und in den Registern des Video Bausteines selbst.

Die Information wie jedes Zeichen aussehen soll, holt sich der C-64 im Zeichensatzmodus aus dem Zeichengenerator, der im Speicher als ROM zwischen D000 – 0FFF (53248 – 57343) abgelegt ist. Er ist 4095 (FFF Hex) Byte lang und enthält zwei Zeichensätze.

Am Anfang des Video Blockes finden wir die Registeradressen für den Video Chip 6566 (D000 – D02E Hex; 53248 – 53294 Dez). Über diese Register lassen sich die Funktionen des Video Bausteines auswählen und steuern. ROM und RAM überlappen sich hier. Der Baustein 6566 arbeitet größtenteils unabhängig von der CPU und unterbricht den 6510 nur ca. alle 0,5 ms um notwendige Befehle zu übernehmen. Wenn die CPU zeitgenau Arbeiten durchführen will und nicht gestört sein will, trennt sie den 6566 Baustein ab (siehe Bildschirm während des Lade- und Speichervorganges). Der Video Baustein kann insgesamt 16K Speicher gleichzeitig adressieren. Über zwei Kontrolleitungen kann der jeweilige 16K Block ausgewählt werden. Ausgewählte Blöcke bleiben solange selektiert, bis wieder auf einen anderen Block umgeschaltet wird. Die Steuerung erfolgt über die zwei niederwertigen Bits der Adresse DD00 (56576).

POKE 53265,11 (Blank) zurücksetzen Poke 53265,27

Nach dem Einschalten teilt sich der Video Chip die 16K innerhalb des angewählten Blockes wiefolgt ein:

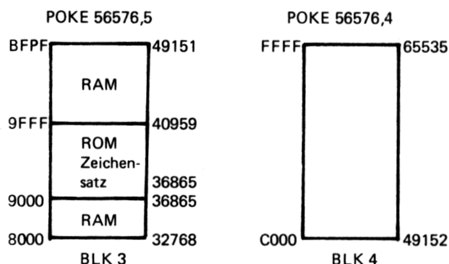


Adressenbereich des 6566 Bausteines nach dem Einschalten.
Diese Konfiguration entspricht POKE 56576,7

Überzeugen Sie sich davon durch PRINT PEEK (56576); im Direktmodus. Neben dieser Kofiguration können Sie durch POKE in die Zelle 56576 noch die drei anderen Blöcke anwählen.

POKE 56576,6 wählt einen Block von 16384 – 32767 aus in diesem Block ist kein Zeichensatz enthalten. Hier könnte man zum Beispiel einen eigenen Zeichensatz einlesen.

POKE 56576,5 bewirkt, daß der Video Baustein den Bereich von 32768 – 49151 auswählt. Der 6560 erkennt RAM von 32768 – 36863 und von 40960 – 49151. Der Zeichensatz liegt von 36864 – 40959.



Bitte beachten Sie, daß die Adressenbelegung vom Video Chip aus gesehen gelten und sich nicht auf den Adressenbereich der CPU beziehen. Der Bildschirmspeicher muß immer in dem jeweiligen Block liegen. Nach dem Einschalten liegt er ab Adresse 1024 Dez (400 Hex). Dies fällt auch gerade in Block 1. Wenn wir auf Block 4 mit POKE 56576,5 umschalten würden, wäre der Bildschirm verschwunden. Sie müssten also vorher den Bildschirm in diesen Bereich verlegen. Wie geht das ? In Zelle 53272 liegt die Anfangsadresse des Bildschirmspeichers und des Zeichensatzes. Beide Adressen können Sie mit dem folgenden Programm auslesen.

```
10 X=PEEK(53272)
20 Y=INT(X/16)*1024
30 PRINT"BILDSCHIRM BEGINNT AN ";Y
40 Z=X-(INT(X/16))*16
50 IF Z/2=INT(Z/2)THEN 70
60 Z=Z-1
70 W=Z*1024
80 PRINT"ZEICHENSATZ BEGINNT BEI ";W
```

Vor dem Verlegen des Bildschirmspeichers muß man sich im Klaren sein welchen Block man anwählen möchte. Wichtig ist, daß man darauf achtet, daß auch ein Zeichensatz im Block vorhanden ist, es sei denn man hat einen eigenen Character Generator in den entsprechenden RAM-Bereich geschrieben. Die Anfangsadresse des Bildschirmes wird in BASIC über die Zelle 648 Dez. festgelegt. Sie kann im jeweils angewählten Block praktisch an jede RAM Adresse gelegt werden, die durch 1024 teilbar ist. Es wäre z. B. möglich BLK 3 anzuwählen und den Bildschirmspeicher ab Adresse 32768 zu legen. Dies ist die Bildschirmadresse des CBM 3032 (4032) Computers. Wir wollen hierfür einmal die entsprechenden Werte ermitteln. Die Formel aus Zeile 20 im vorangegangenen Programm muß als Ergebnis Null liefern, und der Zeichensatz kann an der gleichen Stelle, bezogen auf die Anfangsadresse des Blockes bleiben (nämlich an Adresse 4096). Diese Adresse sowie die Adresse 32768 + 4096 ist eine logische Adresse des VIC-Chips.

	Bildschirm				Zeichensatz			Inhalt Dez	Inhalt	
D018	13	12	11	10	13	12	11	21	53272 Dez	15 Hex
D018	0	0	0	1	0	1	0	1	21	53272 Dez	15 Hex

Inhalt der Adresse 53272 Dez nach dem Einschalten des C-64. Das niederwertigste Bit ist unbedeutend.

Aus dem Registerinhalt der Speicherstelle 53272 sehen wir daß innerhalb des angewählten Blockes die Bildschirmadresse nach dem Einschalten auf 1 steht. Dies bedeutetd, daß die erste durch 1024 teilbare Adresse selektiert wird. Sind in den vier Bit, die dem Bildschirm zugeordnet sind alle 4 Bit auf Null gesetzt, beginnt der Bildschirm auch an Adresse 0 des Blockes. In unserem Falle ist das später im BLK 3 dann die Adresse 32768.

Wir ersehen hieraus, daß unser Bildschirm immer nur an den Adressen beginnen kann, die ein Vielfaches von 1024 betragen (oder auch am Anfang des Blockes, wenn alle zugehörigen Bit = 0 sind). Mit diesen 4 Bits kann ich nämlich im Adressenbereich von 32767 – 16394 = 16383 Dez genau 16 Teilblöcke adressieren. Der Teil, der vom Zeichengenerator belegt wird, darf natürlich nicht als Bildschirmspeicher selektiert werden. Ähnlich verhält es sich mit dem Zeichensatz. Er ist nach dem Einschalten über Bit 2 = gesetzt (selektiert).

0	0	0	1	Bildschirmadressauswahl im Block
	0	1	0	Zeichensatz-Auswahl im Block

Die Situation nach dem Einschalten in Adresse 53272

Die drei Bits erlauben insgesamt 8 verschiedene Anfangsadressen des Zeichensatzes innerhalb eines Blockes zu wählen. Die gesetzte 1 im Bit 2 sagt uns, daß der Zeichensatz bei 16384/8 x 2 = 4096 beginnt. Dies heißt 4096 Bytes nach der Startadresse des ersten Blockes. Dies auf Block drei übertragen heißt, daß wir unserer Register wiefolgt setzen müssen.

8	4	2	1	
0	0	0	0	Bildschirmauswahl in Block 3
	0	1	0	Zeichensatzauswahl in Block 3

Zusammengesetzt ergibt dies für die Adresse 53272

128	64	32	16	8	4	2	1
0	0	0	0	0	1	0	1

POKE 53272,5

Der Bildschirm ist jetzt an die Adresse 32768 Dez verlegt worden und der Zeichensatz ist an seiner alten Stelle, bezogen auf den Blockanfang geblieben. BASIC bekommt jetzt noch den Anfang des Bildschirmes über POKE 648,128 mitgeteilt.

Ab sofort liegt der Bildschirm Ihres C-64 jetzt nicht mehr an der Adresse 1024, sondern an der Adresse 32768 Dez. Wenn Sie jetzt noch PET oder CBM Programme laden wollen, können Sie den BASIC-Anfang des C-64 von 2048 Dez auf 1024 Dez verlegen, indem Sie die BASIC-Zeiger in der Zeropage entsprechend umstellen.

```
POKE 43,0
POKE 44,04
POKE 56,64 (setzt RAM-Ende auf 16K CBM)
POKE 56,128 (setzt RAM-Ende auf 32K CBM)
```

Sie können jetzt einen CBM 16K oder 32K simulieren. Wie Sie sicher wissen dürfen Sie jedoch keine Programme verwenden die Zero Page Adressen oder BASIC Routinen benutzen.

Aus diesen Möglichkeiten, die nur einen sehr kleinen Teil derer darstellen, die der C-64 bietet, erkennen Sie die Leistungsfähigkeit dieses Rechners.

Sie können jeweils einen dieser drei Blöcke des Video Chips anwählen und dort mehrere Bildschirmspeicher und mehrere Zeichensätze de-

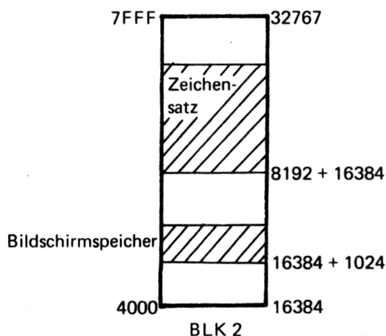
ponieren und im Programm umschalten. Wir wollen uns nun einmal ansehen, wie man einen eigenen Zeichensatz erstellen kann.

Der C-64 hat in Block 1 und 3 einen Zeichensatz ab Adresse 4096 vom jeweiligen Blockanfang an aus gesehen. Er ist jeweils 4K lang und reicht bis zur Adresse 8191 bezogen auf den Blockanfang. Der ROM Zeichensatz ist im Adressbereich der CPU von D000 – DFFF angesiedelt, und kann hier vom Video Chip aus unter den Adressen 4096 – 8191 in Block 1 oder 36864 – 40959 in Block 2 erreicht werden (virtuelle Adressen = logische Adressen ! nicht physikalisch !).

Wir wollen nun, um den Fall etwas interessanter zu machen, den eingebauten Zeichensatz in Block 2 legen. Dort haben wir 16K RAM zur Verfügung, die wir jetzt selbst gestalten wollen. Wie schon angedeutet müssen wir den Bildschirm auch mit in den Block 2 verlegen, da Bildschirm, Zeichensatz und wie wir später noch sehen werden die Sprites immer innerhalb eines Blockes liegen müssen.

Wenn der Computer eingeschaltet wird, ist er über die Adresse 1 so konfiguriert daß man vom Programm her nur den RAM-Bereich über dem Zeichensatz ab Adresse D000 ansprechen kann. Ein wegklappen dieses RAM-Bereiches kann durch setzen der entsprechenden Bit in Adresse 1 erfolgen. Man kann dann den ROM-Bereich unter dem I/O RAM lesen und in den Block 2 schreiben. Dort muß er an eine Stelle gebracht werden, die über die Adresse 53272 festgelegt werden muß. Den Bildschirmspeicher belassen wir wieder an Adresse 1024 + Blockanfang von Block 2.

Zuerst müssen wir die Anfangsadressen des Zeichensatzes und des Bildschirmspeichers in Block 2 festlegen.

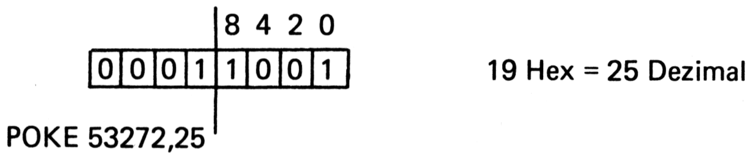


4	2	1
1	0	0

Zeichensatz-Register

4 x 2048 = 8192 Dez.

Die Offsetadresse für den Zeichensatz legen wir an Blockanfang + 8192 = 24576 (6000 Hex).
In Adresse 53272 muß nun folgender Wert gepoked werden.



POKE 53272,25

Damit liegt unser Bildschirmspeicher an Adresse 17408 Dez (4400 Hex) und unser Zeichensatz an Adresse 24576 (6000 Hex). Beide liegen im Block 2 des Adressierungsbereiches unseres Video Bausteines.

Programm zur Verschiebung des Zeichensatzes im RAM

```
1 POKE 56333,127
3 POKE 56,67
4 POKE 1,51
5 Z=24576
10 X=53248
20 Y=PEEK(X):X=X+1
30 IF X=57344 THEN 65
50 POKE Z,Y:Z=Z+1
60 GOTO 20
65 POKE 1,55
70 POKE 56576,6:POKE 53272,25
80 POKE 648,68
90 POKE 56333,129
95 END
```

Programmbeschreibung:

1 POKE 56333,127 Interrupt abschalten
3 POKE 56,67 BASIC Bereich is max. 67 Pages (17K) begrenzen
4 POKE 1,51 I/O über Zeichensatz abschalten, Zeichensatz für PEEK offenlegen
5 Z = 24576 Anfang Zeichensatz neu
10 X = 53248 Anfangsadresse Zeichensatz in ROM ab D000 Hex
20 – 60 Umschaufeln der Werte
65 I/O wieder einschalten

70 Block 3 anwählen und Adressen für Bildschirmspeicher und Zeichensatz festlegen
80 BASIC mitteilen, daß Bildschirmraum jetzt bei Page 68 anfängt
90 Interrupt wiederaktivieren

Unser Zeichensatz ist jetzt identisch, zu dem in ROM, befindet sich aber in RAM ab Adresse 24576 Dez (6000 Hex) und ist leicht erreichbar, da an dieser Stelle der RAM-Bereich nicht durch ROM überlagert ist. Wichtig ist daß wir BASIC nach oben hin begrenzen. POKE 56,67 grenzt BASIC eine Page vor Beginn des Zeichensatzes ab. Wenn wir wollen, können wir den BASIC-Anfang auf 0400 Hex vorverlegen, da wir den Bildschirmspeicher nach 4400 Hex (17400 Dez) verlegt haben.

Interessant ist, daß unser Zeichensatz jetzt ab Adresse 6000 Hex in RAM. Die Adresse 6000 Hex ist jetzt im Gegensatz zu früher nicht nur eine logische Adresse, sondern physikalische und logische Adresse zugleich. Wie wir wissen ist der Adressbereich des Zeichensatzes in Block 1 und 3 nur logisch. D. h. zwischen den Adressen 4096 und 8191 im ersten Block und den Adressen 36865 und 40959 im Block 3 liegen die Adressen nur logisch an. Der Videobaustein verhält sich so, als ob die Adressen des Zeichensatzes in diesem Bereiche liegen würden. In Wirklichkeit liegt der Zeichensatz jedoch in ROM zwischen den Adressen D000 und DFFF Hex. Ist der Standardzeichensatz angewählt, und befindet sich dieser in ROM zwischen D000 – DFFF so sind die Adressen in den beiden Blöcken nur logische Adressen. Wir wissen ja auch, daß wir nach dem Einschalten den Bereich 4096 – 8191 Dez für unsere BASIC-Programme verwenden können. Verlegen wir jetzt unseren Zeichensatz an eine andere Stelle im gleichen Block oder in einen anderen Block in den RAM Bereich, so wird die logische Adresse im Block auch gleichzeitig zur physikalischen Adresse. Wie in unserem kleinen Beispiel. Hier liegt der Zeichensatz jetzt ab 6000 Hex. Es wäre in der Praxis natürlich sinnvoller man würde einen geänderten Zeichensatz in den RAM-Bereich unter ROM-BASIC legen und bräuchte so kein Arbeitsspeicherteil zu opfern.

Der Zeichensatz kann nun von uns nach Belieben geändert werden.

Untersuchung des Zeichengenerators

Wenn wir den Zeichengenerator mit dem vorliegenden Programm nach

6000 Hex (24576) gebracht haben können wir diesen in Ruhe ohne I/O-Abschaltung oder Interruptunterbrechung untersuchen und ändern.

Programm zur Untersuchung von Zeichen

Auch dieses Programm lässt sich nur bei einem in den RAM-Bereich verschobenen Zeichensatz verwenden. Die Eingaben:

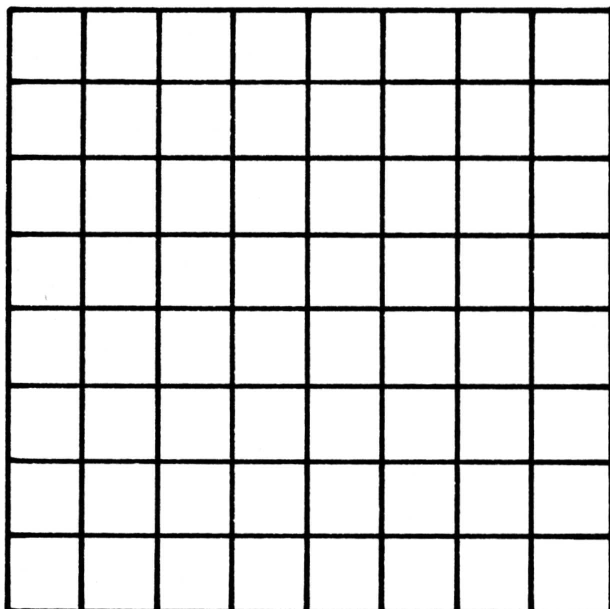
Anfangsadresse	24576	(Das Programm finden Sie auf
Endadresse	24583	der übernächsten Seite)

liefern uns den Aufbau des ersten Zeichens: 60, 102, 110, 96, 98, 60, 0

Hex	Binär								Dezimal	Adressen
3C			●	●	●	●			60	24576
66		●	●			●	●		102	24577
6E		●	●		●		●	●	110	24578
6E		●	●		●		●	●	110	24579
60		●	●						96	24580
62		●	●				●		98	24581
3C			●	●	●	●			60	24582
0									0	24583

Überall wo ein Punkt ist, ist das Bit gesetzt, alles andere ist Null. Aus den Hex-Werten lässt sich das Bild schnell ermitteln. Man teilt das Raster in zwei Hälften und trägt dann in beide Hälften immer die einstelligen Hexwerte ein. Z. B. 3C ergibt eine 3 in der linken Hälfte und C in der rechten Hälfte 3 = 0 0 1 1, C = 1 1 0 0. Damit erhalten Sie die

oberste Zeile. Wenn wir unser kleines Programm wieder mit RUN 400 starten und 24584 als Anfangsadresse und 24591 als Endadresse eingeben, erhalten wir die Zusammensetzung des Buchstabens A. Zwischen den Buchstaben erkennen wir eine Null *als* Trennelement. Auf diese Weise können wir jetzt ein Programm zur Änderung des Zeichensatzes leicht entwerfen.



Das erste Zeichen 0 für das PRINT AT Zeichen (Klammeraffe) liegt sofort am Anfang des Zeichensatzes. Es belegt 8 Bytes zu je 8 Bit woraus wir schon den Aufbau eines einzelnen Zeichens haben. Wenn wir später selbst Zeichen verändern wollen, so müssen wir den Aufbau kennen.

Nachfolgend ein kleines Programm, welches Ihnen die Zeichen aus dem Zeichensatz auf den Bildschirm ausgibt. Sie können sich an Hand dieses Programmes überzeugen, daß unser verschobener Zeichensatz nun auch wirklich an der Adresse 24576 liegt.

```
200 INPUT"ZAHL FUER DAS ZEICHEN";Z
210 IF Z<0 ORZ>255 THEN 310
220 X=24576+8*Z
230 FOR N=1 TO 8
240 Y=PEEK(X+N)/128
250 FOR T=1 TO 8
260 Y%=Y:Y=(Y-Y%)*2
270 PRINT CHR$(32+Y%*3);
280 NEXT T
290 PRINT
295 NEXT N
300 GOTO 200
310 END
```

Programm zur Untersuchung des Zeichensatzes. Achtung, dieses Programm erlaubt nur die Auslistung der Zeichen, wenn der Zeichensatz, wie vorher beschrieben, an die Adresse 6000 Hex verschoben wurde. Die Zahlen für die Eingabe entnehmen Sie bitte dem C-64 Handbuch auf Seite 133 (Bildschirmcodes).

```
400 INPUT "ANFANGSADRESSE";A
410 INPUT "ENDADRESSE";B
420 FOR X=A TO B
430 PRINT PEEK(X);
440 NEXT X
```

Wer Änderungen im Zeichensatz machen möchte, sollte sich dann auf jeden Fall die Zusammensetzung des Zeichens mit der zugehörigen Adresse auf einem Drucker ausdrucken und dann entsprechend ändern.

Hier noch ein kleines Programm, für Sie, wenn Sie nach dem Einschalten des C-64 den Zeichensatz ansehen wollen. D. h. wenn der Zeichensatz an der physikalischen Adresse D000 und an der logischen Adresse 4096 aktiviert ist.

Geben Sie die Anfangsadresse 53248 und 53255 als Endadresse ein und Sie erhalten den Aufbau des ersten Zeichens (@), wie wir ihn vorher bereits besprochen haben. Nur haben wir das Zeichen da aus dem ver-

schobenen Zeichensatz herausgelesen.

```
480 INPUT"ANFANGSADRESSE";A
490 INPUT"ENDADRESSE";B
500 POKE 56333,127
510 POKE 1,51
515 T=0
520 FOR X=A TO B
530 Y(T)=PEEK(X)
531 T=T+1
540 NEXT X
550 POKE 1,55
560 POKE 56333,129
570 FOR T=0 TO 7
580 PRINT Y(T);
590 NEXT T
600 END
```

Um den Zeichensatz selbst abändern zu können, muß man zuerst die Adresse aufsuchen, an der sich das Zeichen befindet. Dann kann man sich die 8 x 8 Matrix ansehen und im RAM ändern. Ein Zeichen, welches man kaum benötigt könnte man dann z. B. in ein Umlaut ü usw. abändern. Den geänderten Zeichensatz mit den zugehörigen POKE Befehlen könnte man dann auf Cassette oder Diskette speichern und bei Bedarf einlesen.

Das Verständnis der Zusammenhänge um die einzelnen Zeichen und den Zeichensatz sind insofern wichtig, da sie später den Aufbau der Sprites leichter verstehen werden.

Der Tonausgabebaustein im C-64

Der SID (Sound Interface Baustein) aus Commodores eigener Halbleiterproduktion beinhaltet einen Synthesizer zur Erzeugung von dreistimmigen Toneffekten.

Pro Oszillatorausgang, also pro Stimme können Sie vier verschiedene Wellenformen erzeugen.

1. Dreieck
2. Sägezahn
3. Rauschen
4. Rechteck mit veränderlichem Tastverhältnis

Für jeden einzelnen Kanal können Sie Hüllkurve, Frequenz, Tastverhältnis und Wellenform, getrennt programmieren.

Drei Hüllkurvengeneratoren und ein Amplitudenmodulator erlauben die Generierung unterschiedlichster Laute und Töne. Weiterhin sind ein programmierbarer Filter, Lautstärkenregelung und A/D-Wandler für den Anschluß von Paddles oder einfach Datenaquisition eingebaut. Wir wollen hier nicht den Baustein besprechen, sondern wieder einige Beispiele geben, wie Sie Ihre Programme durch zusätzliche Tonausgabe erweitern können. Wer tiefer einsteigen möchte, dem sei das Datenblatt des 6581 SID empfohlen. Auch das Reference Manual für den C-64 erhält mehr detailierte Informationen.

Registeraufbau des SID 6581

Die Register des SID-6581 finden wir ab Adresse D400 (54272) in RAM. Dies ist, ähnlich wie beim Video Controller Baustein, der Bereich über dem Zeichengenerator in ROM. Der SID enthält insgesamt 29 Steuerregister zur Erzeugung von Tönen. Alle Register können entweder nur gelesen oder nur beschrieben werden.

Die ersten beiden Register

54272 (D400)
74273 (D401)



bilden zusammen ein 16 Bit Register, mit dem die Tonhöhe (Frequenz) von Oszillator 1 eingestellt werden kann. Die Tonfrequenz kann wiefolgt berechnet werden.

F = (Zahl im Register X 0.059604645) Hz

Auf Seite 158 des mitgelieferten Handbuches finden Sie eine Frequenztabelle.

Die zwei folgenden Register

54274 (D402)
54275 (D403)



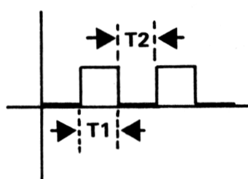
bestimmen das Tastverhältnis des Tones (Impuls/Pause). Sie bilden zusammen ein 13 Bit Register, wobei die Bits 4 – 7 nicht verwendet werden.

TV = (Zahl im Register: 40.95) %

Das Tastverhältnis in % errrechnet sich aus der Zahl im Register geteilt durch 40.95. Ein Wert 0 oder FFF in diesem Register erzeugt keinen hörbaren Ton. Deshalb muß hier immer ein Wert eingegeben werden. Die Werte 2048 ($800) dagegen erzeugen einen Impuls mit dem Tast-

verhältnis 50%.

Z. B.



$$\text{Tastverhältnis TV} = \frac{T1}{T2}$$

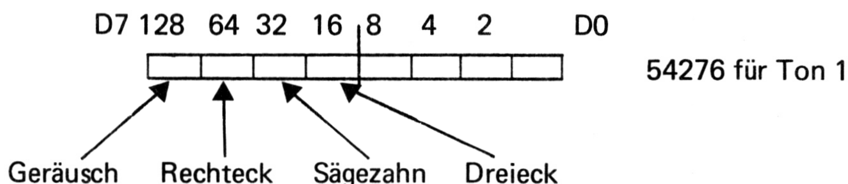
Da wir drei Töne gleichzeitig erzeugen können und jeder Ton die vorher beschriebenen Register benötigt, sind schon 12 Adressen verbraucht. Wir können jetzt Frequenz und Tastverhältnis festlegen.

Die Adressen liegen wiefolgt (Frequenzadressen und Tastverhältnis-adressen):

1. TON	54272 + 54273	(D400 + D401)
1. TV	54274 + 54275	(D402 + D403)
2. TON	54279 + 54280	(D407 + D408)
2. TV	54281 + 54282	(D409 + D40A)
3. TON	54286 + 54287	(D40E + D40F)
3. TV	54288 + 54289	(D410 + D411)

Wenn Sie Frequenz und Tastverhältnis festgelegt haben, müssen Sie in den Registern 54276, 54283 und 74290 die gewünschte Wellenform festlegen.

54276 = für TON 1 (D404)
54283 = für TON 2 (D40B)
54290 = für TON 3 (D412)

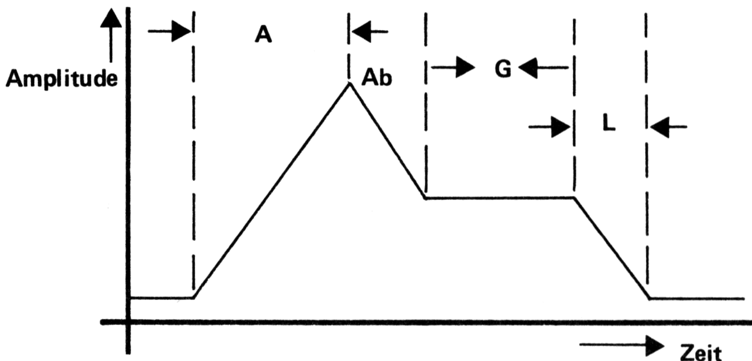


64 Dez entspricht z. B. einem Rechteckgenerator.

Die ersten vier Bit (D0 – D4) dienen auch zur Beeinflussung des Tones. D0 steuert den Hüllkurvengenerator für den Ton 1. Wenn Bit 1 gesetzt ist, müssen auch die Hüllkurvenparameter für die drei Töne gesetzt werden. Die Hüllkurven können in folgenden Registern für die einzelnen Töne gesetzt werden:

	Adresse Dez			Adresse Hex
Ton 1	54277	Ansteigende Hüllkurve	Abfallende Hüllkurve	D405
Ton 1	54278	gleichbleibend	langsam abklingend	D406
Ton 2	54284	Ansteigende Hüllkurve	Abfallende Hüllkurve	D40C
Ton 2	54285	gleichbleibend	langsam abklingend	D40D
Ton 3	54291	Ansteigende Hüllkurve	Abfallende Hüllkurve	D413
Ton 3	54292	gleichbleibend	langsam abklingend	D414

Mit diesen Registern kann jetzt für jeden der drei Töne die Hüllkurve individuell bestimmt werden. Zuerst wird Bit 0 (D0) des zugehörigen Registers gesetzt. Z. B. 54276 für Ton 1.



Im Bild sehen Sie die einzelnen Hüllkurvenwerte

A = Ansteigende Hüllkurve (Attack)
Ab = Abfallende Hüllkurve (Decay)
G = Gleichbleibende Hüllkurve (Sustain)
L = Langsam ausklingend (Release)

Durch die Programmierung der jeweiligen Halbregister kann ich die Zeit für die einzelnen Hüllkurventeile festlegen. Es stehen jeweils 4 Bit für jeden Hüllkurventeil pro Ton zur Verfügung. Aus der 4 Bit Auflösung (16 Möglichkeiten) können Sie sich die ungefähren Zeiten pro Bitmuster ausrechnen, indem Sie die Bereiche durch 16 dividieren.

Ansteigende Hüllkurve 2ms – 8 Sek.
Abfallende Hüllkurve 6ms – 24 Sek.
langsam ausklingend 6ms – 24 Sek.

Die gleichbleibende Hüllkurve lässt sich auch in 16 Schritten einstellen. Die Amplitude bezieht sich dann auf den Wert der letzten abfallenden oder ansteigenden Hüllkurve. Ein Mittelwert von 8 in D4 – D7 des Registers 54278 und eine abfallende Hüllkurve in 54277 würde die abfallende Hüllkurve auf den halben Wert sinken lassen und dort belassen, bis D0 von 54276 wieder auf Null gesetzt wird.

Wir wollen nun etwas experimentieren und unsere ersten Töne erzeugen. Der SID 6581 kann Frequenzen bis 4 KHz erzeugen. Dies bedeutet, daß wir mit einem 16 Bit Register den Bereich 0 – 4000 Hz in 65536 Schritten von 0 – 65536 verändern können. Man könnte sich jetzt ein kleines BASIC-Programm schreiben, welches Ihnen die Werte für die verschiedenen Töne ausrechnen und auflisten kann.

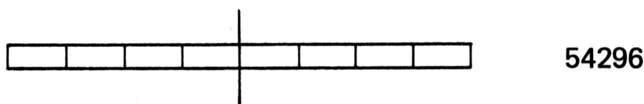
Nach unserer vorher gezeigten Formel, müssen wir für einen Ton von 1000 Hz ca. folgenden Wert in die beiden Register bringen.

$$X = \frac{1000}{0.059604645}$$

X = 16777 dez
X = 4189 hex

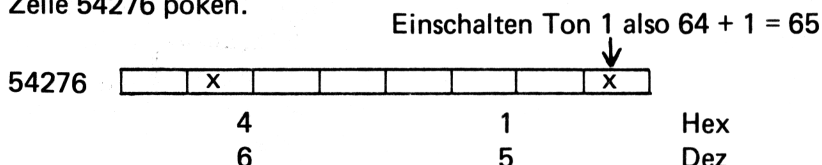
Aufgeteilt in zwei Hex Bytes ergibt 41 und 89 und daraus die Dezimalwerte 65 und 137.

Die Lautstärke wird immer gleichzeitig für alle drei Tonkanäle eingestellt. Das zugehörige Register finden wir in den unteren vier Bit der Zelle 54296 (D418).



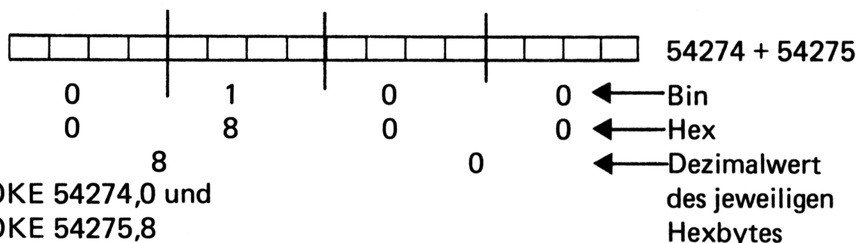
Wie wir sehen kann die Lautstärke über diese vier Bit in 16 Stufen von 0 – 15 eingestellt werden. In dieses Register kann auch nur hineingeschrieben werden. Wir können durch ? PEEK nicht nachsehen, welche Lautstärke gerade gewählt wurde. Mit POKE 54296,0 werden alle Tonausgänge abgeschaltet.

Die Einstellung der Wellenform ist auch einfach. Wir haben sie bereits besprochen. Wenn wir einen Rechteckgenerator wollen, müssen wir 64 in Zelle 54276 poken.



54276 für Ton 1
54283 für Ton 2
54290 für Ton 3

Achtung !
Das letzte Bit in 54276 schaltet den Hüllkurvengenerator ein. Das Tastverhältnis soll 50% sein. Es wird in den ersten 12 Bit der Register 54274 und 54275 festgelegt.



POKE 54274,0 und
POKE 54275,8

Nun müssen wir noch die Hüllkurve festlegen. Eine ansteigende Hüllkurve und Dauterton.

```
100 POKE 54296,15:REM LAUTSTAERKE
110 POKE 54277,128:REM ANSTEIGENDE HULL
120 POKE 54278,128:REM GLEICHBLEIB
130 POKE 54272,137:REM FREQUENZ REG1
140 POKE 54273,65  :REM FREQUENZ REG2
150 POKE54274,0:REM TASTVERH REG1
160 POKE54275,8:REM TASTVERH REG2
170 POKE54276,65:REM WELLENFORM
```

Sound 1

Das vorherige Programm soll unser Beispiel sein. Woher die einzelnen Werte kommen, ist bereits beschrieben worden. Es erzeugt auf Tonkanal 1 einen ca. 1 KHz Rechteckton. Es wird mit RUN gestartet und mit POKE 54276,64 kann der Ton wieder abgeschaltet werden.

Die Dauer des Tones können Sie durch eine Schleife vor dem Abschalten bestimmen.

```
175 FOR X = 1 TO 2000 : NEXT X
```

Wir haben jetzt gesehen wie man einen Ton bestimmter Frequenz und mit einer gewünschten Hüllkurve ausgeben kann. Alle anderen Möglichkeiten bauen jetzt darauf auf. Sie können jetzt einen zweiten und dritten Ton unterschiedlicher Frequenz erzeugen und mischen. Vielleicht sollte man noch einige Versuche mit verschiedenen Frequenen auf mehreren Kanälen durchführen.

Eine zweite Frequenz etwas abweichend auf Kanal mit gleicher Hüllkurve und gleichem Tastverhältnis ergibt folgendes Programm:

```
100 POKE 54296,15:REM LAUTSTAERKE
110 POKE 54277,128:REM ANSTEIGENDE HULL
120 POKE 54278,128:REM GLEICHBLEIB
130 POKE 54272,137:REM FREQUENZ REG1
```

```
140 POKE 54273,65 :REM FREQUENZ REG2
150 POKE54274,0:REM TASTVERH REG1
160 POKE54275,8:REM TASTVERH REG2
170 POKE54276,65:REM WELLENFORM
175 FOR X=1 TO 2000:NEXT X
180 POKE 54276,64
200 POKE 54284,128:REM ANSTEIGENDE HULL
300 POKE 54285,128:REM GLEICHBLEIB
400 POKE 54279,100:REM FREQUENZ REG1
500 POKE 54280,40 :REM FREQUENZ REG2
600 POKE54281,0:REM TASTVERH REG1
700 POKE54282,8:REM TASTVERH REG2
800 POKE54283,65:REM WELLENFORM
900 FOR X=1 TO 2000:NEXT X
950 POKE 54283,64
```

Sound 2

Ab Zeile 200 finden Sie den zweiten Ton. Sie können jetzt nach belieben selbst Hüllkurven, Frequenz, Tastverhältnis, usw. verändern und eigene interessante Versuche anstellen. Sie können der Einfachheit halber die Zeilen leicht mit dem Cursor duplizieren, indem Sie einfach mit dem Cursor über die Zeilennummer das Programm für Tonkanal 1 fahren und einfach die neuen Zeilennummern darüber schreiben.

Geben Sie dann List ein und ändern die Register für Kanal 2 ab. Ändern Sie auch die Werte für Frequenz, Tastverhältnis usw. je nach Bedarf. Auf diese Weise sparen Sie sich viel Arbeit an der Tastatur.

Wer sich eingehender mit Tonerzeugung beschäftigen will, für den gibt es beim Hofacker Verlag ein Buch "Musical Applications of micro processors" ca. 650 Seiten, DM 79,–.

Beispiel für dreistimmige Tonerzeugung

Das nachfolgende Programm steuert alle drei Tongeneratoren. Der Ton jedes einzelnen Kanales, ist er einmal aktiviert, bleibt so lange, bis er über das zugehörige Register wieder abgeschaltet wird.

```
100 POKE 54296,15:REM LAUTSTAERKE
110 POKE 54277,128:REM ANSTEIGENDE HULL
120 POKE 54278,128:REM GLEICHBLEIB
130 POKE 54272,137:REM FREQUENZ REG1
140 POKE 54273,65  :REM FREQUENZ REG2
150 POKE54274,0:REM TASTVERH REG1
160 POKE54275,8:REM TASTVERH REG2
170 POKE54276,65:REM WELLENFORM
210 POKE 54284,128:REM ANSTEIGENDE HULL
220 POKE 54285,128:REM GLEICHBLEIB
230 POKE 54279,100:REM FREQUENZ REG1
240 POKE 54280,40  :REM FREQUENZ REG2
250 POKE54281,0:REM TASTVERH REG1
260 POKE54282,8:REM TASTVERH REG2
270 POKE54283,65:REM WELLENFORM
300 POKE 54291,128:REM ANSTEIGENDE HULL
320 POKE 54292,128:REM GLEICHBLEIB
330 POKE 54286,180:REM FREQUENZ REG1
340 POKE 54287,70  :REM FREQUENZ REG2
350 POKE54288,0:REM TASTVERH REG1
360 POKE54289,8:REM TASTVERH REG2
370 POKE54290,65:REM WELLENFORM
380 FOR X=1 TO 2000:NEXT X
390 POKE 54290,64
400 POKE54276,64:POKE 54283,64
```

Sound 3

Einfache Sound Beispiele

RANDOM Noise

Dieses Programm erzeugt Zufallstöne über 2 Kanäle. Es muß mit RUN 400 gestartet werden !

```
100 POKE 54296,L:REM LAUTSTAERKE
110 POKE 54277,128:REM ANSTEIGENDE HULL
120 POKE 54278,128:RETURN
130 POKE 54272,P
140 POKE 54273,Q
150 POKE54274,0:REM TASTVERH REG1
160 POKE54275,8:REM TASTVERH REG2
170 POKE54276,65:RETURN
```

```
200 POKE 54296,L:REM LAUTSTAERKE
210 POKE 54284,128:REM ANSTEIGENDE HULL
220 POKE 54285,128:RETURN
230 POKE 54279,P
240 POKE 54280,Q
250 POKE54281,0:REM TASTVERH REG1
260 POKE54282,8:REM TASTVERH REG2
270 POKE54283,65:RETURN
400 FOR P=5 TO 30  STEP RND(1)*15+0.2
410 Q=P
420 GOSUB 130
430 L=(RND(1)*10+5)/(0.1*P)
440 GOSUB 100
450 GOSUB 200
460 GOSUB 230
470 NEXT P
480 POKE54276,64:POKE54283,64
```

Motorengeräusch

Achtung !
Dieses Programm mit RUN 400 starten !

```
100 POKE 54296,L:REM LAUTSTAERKE
130 POKE 54272,P
140 POKE 54273,Q
150 POKE54274,0:REM TASTVERH REG1
160 POKE54275,8:REM TASTVERH REG2
170 POKE54276,65:RETURN
400 L=15
410 GOSUB 100
420 P=0:Q=8
430 GOSUB 130
450 POKE 54276,64
470 GOTO 400
```

Musik Stück auf C-64

```
0 PRINT"♥"+CHR$(14)+CHR$(158):POKE53280,6:POKE53281,11:TI$
="000000"
1 PRINT"▒▒▒▒▒▒DER DRITTE MANN"
8 PRINT"▒▒▒▒▒BITTE WARTEN SIE 3 MIN."
9 PRINT"▒UND 25 SEKUNDEN ES WIRD EIN FELD GELADEN"
10 S=54272:FORL=STOS+24:POKEL,0:NEXT
20 DIMH%(2,1200),L%(2,1200),C%(2,1200)
30 DIMFQ(11)
40 V(0)=17:V(1)=65:V(2)=33
45 POKES+3,8
50 POKES+9,140:POKES+22,240:POKES+23,244
60 FORI=0TO11:READFQ(I):NEXT
100 FORK=0TO2
110 I=0
120 READNM
121 PRINTTI$:PRINT"♥";
130 IFNM=0THEN250
140 WA=V(K):IFNM<0THENNM=-NM:WA=1
150 DR%=NM/128:OC%=(NM-128*DR%)/16
160 NT=NM-128*DR%-16*OC%
170 FR=FQ(NT)
180 IFOC%=7THEN200
190 FORJ=6TOOC%STEP-1:FR=FR/2:NEXT
200 HF%=FR/256:LF%=FR-HF%*256
210 IFDR%=1THENH%(K,I)=HF%:L%(K,I)=LF%:C%(K,I)=WA:I=I+1:GO
TO120
220 FORJ=1TODR%-1:H%(K,I)=HF%:L%(K,I)=LF%:C%(K,I)=WA:I=I+1
:NEXT
230 H%(K,I)=HF%:L%(K,I)=LF%:C%(K,I)=WA-1
240 I=I+1:GOTO120
250 IFI>IMTHENIM=I
260 NEXT
300 PRINT"♥IIII";
302 PRINT"                                        "
303 PRINT"                                        "
304 PRINT"                                        "
500 POKES+5,63:POKES+6,240
505 POKES+10,8
510 POKES+12,8:POKES+13,9
520 POKES+19,119:POKES+20,7
530 POKES+24,31
```

```
540 FORI=0TOIM
550 POKES,L%(0,I):POKES+7,L%(1,I):POKES+14,L%(2,I)
560 POKES+1,H%(0,I):POKES+8,H%(1,I):POKES+15,H%(2,I)
570 POKES+4,C%(0,I):POKES+11,C%(1,I):POKES+18,C%(2,I)
571 IFI=313THENPOKES+12,119:POKES+13,53
572 IFI=571THENPOKES+12,53:POKES+13,53:POKES+10,6
573 IFI=823THENPOKES+12,10:POKES+13,167:POKES+10,0:POKES+2
0,240:POKES+5,15
580 NEXTI
582 PRINT"BEL.TASTE F.WIEDERHOLUNG"
590 GETA$:IFA$=""THEN590
591 IFA$="▒"THENEND
598 PRINT"⌉                    ⌉"
599 GOTO500
600 DATA35115,37203,39415,41759
610 DATA44242,46873,49660,52613
620 DATA55741,59056,62567,66288
1000 DATA 338,340,336,585,331,583,322,324,320,569
1010 DATA 315,567,306,308,304,553,299,297,296,551,-496,599

1020 DATA 322,323,324,592,324,592,324,1616
1030 DATA 352,354,355,356,352,354,612,347,610,1632
1040 DATA 322,323,324,592,324,592,324,1872
1050 DATA345,343,342,345,352,612,354,352,345,1634
1070 DATA 322,323,324,592,324,592,324,1616
1080 DATA 352,354,355,356,352,354,612,347,610,1632
1090 DATA352,354,356,352,354,612,352,354,352,356,352,354,6
12,352,354,352
1100 DATA356,352,354,612,347,354,1760
1110 DATA340,341,342,599,345,599,340,341,342,599,345,599
1120 DATA340,336,327,329,331,336,338,340,338,336,338
1130 DATA327,340,341,343,345,343,340,341,599,345,599,340,3
41,342,599,345,599
1140 DATA343,345,346,347,603,603,345,342,338,1367
1150 DATA340,341,342,599,345,599,340,341,342,599,345,599
1160 DATA340,336,327,329,331,336,338,340,338,336,338
1170 DATA 1360 , 327 , 326 , 327 , 592 , 329 , 592 , 329 ,
 336 , 329
1180 DATA 327,336,340,599,340,336,327,585,592,340,594,1616
,-448
1190 DATA345,344,1113,608,2146
1200 DATA 341,340,1109,601,1882,343,594,343,594,343,594,11
04,1109
```

```
1210 DATA340,344,347,612,354,347,352,1113,1114,345,344,111
3,608,2146
1220 DATA341,340,1109,601,1882,343,594,343,594,343,594,110
4,853,341
1225 DATA 345,608,599,336,338,340,597,-480,613,-480
1230 DATA592,329,592,329,336,329,327,336,340,599,340,336,3
27
1240 DATA585,592,340,594,1360,608,-416
1250 DATA581,324,581,324,581,-256,329,338,329,336,338,336,
324
1260 DATA583,326,583,326,583,-256,336,340,336,338,340,338,
336
1270 DATA594,337,594,337,594,-256,341,345,341,343,345,343,
341
1280 DATA352,352,1120,601,599,327,327,583,583
1290 DATA581,324,581,324,581,-256,329,338,329,336,338,336,
324
1300 DATA583,326,583,326,583,-256,336,340,336,338,340,338,
336
1310 DATA329,328,329,599,597,336,340,339,340,601,352,343,3
40
1320 DATA592,592,340,594,1360,608,-416
1999 DATA 0
2000 DATA 338,340,336,585,331,583,322,324,320,569
2010 DATA 315,567,306,308,304,553,299,297,296,551,-496,599
,571
2020 DATA 560,567,567,567,565,576,564,576,564,576,551,567,
551,576,567
2030 DATA 571,560,567,567,567,565,576,564
2040 DATA563,562,566,562,569,571,567,569,571
2050 DATA 560,567,567,567,565,576,564,576,564,576,551,567,
551,576,567
2060 DATA-496,576,580,570,580,569,581,568,581
2070 DATA567,580,567,571,576,567,560
2110 DATA 336,338,339,596,340,596,336,338,339,596,340,596,
340,336,327
2115 DATA 1093,1096,1092,1095
2120 DATA 596,340,596,336,338,339,596,340,596
2130 DATA343,345,346,343,599,598,345,336,338
2140 DATA 1355,336,338,339,596,340,596,336,338,339,596,340
,596,340,336,327
2150 DATA 1093,1096,1092,1098
2160 DATA 581,581,579,579,576,576,576,576
```

```
2170 DATA 582,582,325,581,1604,-448
2180 DATA341,342,1109,601,2138
2190 DATA338,340,1106,594,1879,343,570,578,567,568,569,581
,562,581
2200 DATA564,578,568,578,1092,580,580,341,344,1109,597,213
8,338,340,1106,594
2210 DATA1879,343,570,578,567,568,313,309,308,306,840,331
2220 DATA336,592,842,586,585,-480,597,-480
2230 DATA569,569,579,579,580,576,580,576,582,582,325,581,3
24,1092,592,-432
2240 DATA578,321,578,321,578,-256,329,325,329,336,338,336,
329
2250 DATA580,323,580,323,580,-256,336,343,336,338,340,338,
336
2260 DATA587,330,587,330,587,-256,341,345,341,343,345,343,
341
2270 DATA336,336,1104,592,592,324,324,580,580
2280 DATA578,321,578,321,578,-256,329,325,329,336,338,336,
329
2290 DATA580,323,580,323,580,-256,336,343,336,338,340,338,
336
2300 DATA329,328,329,585,585,329,576,580,579,580
2310 DATA582,582,325,581,324,576,567,560,-464
2999 DATA 0
3000 DATA 338,340,336,585,331,583,322,324,320,569
3010 DATA 315,567,306,308,304,553,299,297,296,551,-496,567
,583
3020 DATA560,564,551,570,549,569,548,567,551,564,551,565,5
44,564,564
3030 DATA 583,560,564,551,570,549,569,548
3040 DATA547,546,562,562,566,567,551,553,555
3050 DATA560,564,551,570,549,569,548,567,551,564,551,565,5
44,564,564
3060 DATA-496,560,567,554,567,553,569,552,568,551
3070 DATA567,551,567,560,551,544
3110 DATA340,341,342
3120 DATA560,567,551,567,560,567,551,567,549,569,581,568,5
64,567,551,567,560
3130 DATA567,551,567
3140 DATA 560,567,564,563,562,567,562,569
3150 DATA 567,565,564,562,560,567,551,567
3160 DATA 560,567,551,567,549,569,565,568
3170 DATA 564,567,560,570
```

```
3180 DATA 565,565,566,566,567,567,567,567
3190 DATA 562,562,567,567,560,551,544,-192
3200 DATA549,569,560,569,554,330,329,330,336,594
3210 DATA546,569,553,569,551,570,562,570,554,570,551,552,5
53,569,562,569
3220 DATA564,571,568,571,1081,567,560,549,569,560,569,554,
570,565,570
3230 DATA546,569,553,569,551,570,562,570,554,570,551,552,2
97,293,292,290,1057
3240 DATA560,569,560,560,565,-480,549,-480
3250 DATA565,565,566,566,567,567,576,576,562,562,567,567,1
072,544,-480
3260 DATA549,565,553,565,549,565,553,565,560,564,551,564,5
60,564,551,564
3270 DATA551,565,555,565,551,565,555,565,563,1075,563,564,
-1536
3280 DATA549,565,553,565,549,565,553,565,560,564,551,564,5
60,564,551,564
3290 DATA549,546,548,549,551,567,566,567,553,546,551,555,5
60,551,544,-480
3999 DATA 0
4999 DIM NV(32)
5000 INPUT"NOTE DPO";ND,NP,NO
5005 IFND=0THEN5070
5030 NV=128*ND+16*NO+NP
5040 NV(I)=NV
5050 I=I+1
5060 GOTO5000
5070 FORJ=0TOI-1
5075 A$=STR$(NV(J)):IFSGN(NV(J))=-1THEN5080
5076 A$=RIGHT$(A$,LEN(A$)-1)
5080 PRINTA$+",";
5090 NEXT
```

US-Flagge auf C-64

```
5 POKE53280,0:POKE53281,0: REM SETZT RAND UND HINTERGRUND
10 PRINT"[CLR][HOME]":REM LOESCHT BILDSCHIRM UND CURS-HO
15 REM AMERICAN FLAG
30 PRINT:PRINT:PRINT:PRINT:PRINT
35 FOR K=1 TO 3
40 PRINT TAB(3);CHR$(31);
45 FOR I=1 TO 12
50 PRINT"[RVS] ";
55 NEXT I
60 PRINT CHR$(28);
65 FOR I=1 TO 23
70 PRINT "[RVS] ";
75 NEXT I
80 PRINT
85 PRINT TAB(3);CHR$(31);
90 FOR I=1 TO 12
95 PRINT "[RVS] ";
100 NEXT I
105 PRINT CHR$(5);
110 FOR I=1 TO 23
115 PRINT "[RVS] ";
120 NEXT I
125 PRINT
130 NEXT K
135 PRINT TAB(3);CHR$(31);
140 FOR I=1 TO 12
145 PRINT "[RVS] ";
150 NEXT I
155 PRINT CHR$(28);
160 FOR I=1 TO 23
165 PRINT "[RVS] ";
170 NEXT I
175 PRINT
180 FOR J=1 TO 3
185 PRINT TAB(3);CHR$(5);
190 FOR I=1 TO 35
195 PRINT "[RVS] ";
200 NEXT I
205 PRINT
210 PRINT TAB(3);CHR$(28);
```

```
215 FOR I=1 TO 35
220 PRINT "■ ";
225 NEXT I
230 PRINT
235 NEXT J
240 REM
250 FOR L=54272 TO 54296:POKE L,0:NEXT:REM CLEARS SOUND CH
IP
255 V=54296:W=54276:A=54277:HF=54273:LF=54272:S=54278:PH=5
4275:PL=54274
260 POKE V,15:POKE A,0:POKE PH,0:POKE PL,0
265 POKE S,240:POKE W,17
270 READ H:IF H=-1 THEN POKE W,0:GOTO 320
275 READ L
280 READ D
285 POKE HF,H:POKE LF,L
290 FOR T=1 TO D:NEXT:POKE W,16
295 FOR T=1 TO 50: NEXT
300 GOTO 260
320 GET A$:IF A$="" THEN 320
Y KEY
330 PRINT"■■"
340 PRINT CHR$(5)
345 END
400 DATA 25,177,187,21,154,62,17,37,250,21,154,250,25,177,
250,34,75,500
405 DATA 43,52,187,38,126,62,34,75,250,21,154,250,24,63,25
0,25,177,500
410 DATA 25,177,187,25,177,62,43,52,375,38,126,125,34,75,2
50,32,94,500
415 DATA 28,214,125,32,34,125,34,75,250,34,75,250,25,177,2
50,21,154,250
420 DATA 17,37,250
430 DATA 25,177,187,21,154,62,17,37,250,21,154,250,25,177,
250,34,75,500
435 DATA 43,52,187,38,126,62,34,75,250,21,154,250,24,63,25
0,25,177,500
440 DATA 25,177,187,25,177,62,43,52,375,38,126,125,34,75,2
50,32,94,500
445 DATA 28,214,125,32,94,125,34,75,250,34,75,250,25,177,2
50,21,154,250
450 DATA 17,37,250
455 DATA 43,52,62,43,52,62,43,52,250,45,198,250,51,97,250,
```

```
51,97,500
460 DATA 45,198,125,43,52,125,38,126,250,43,52,250,45,198,
250,45,198,500
465 DATA 45,198,250,43,52,375,38,126,125,34,75,250,32,94,5
00,28,214,125
470 DATA 32,94,125,34,75,250,21,154,250,24,63,250,25,177,5
00,25,177,250
475 DATA 34,75,250,34,75,250,34,75,125,32,94,125,28,214,25
0,28,214,250
480 DATA 28,214,250,38,126,250,45,198,125,43,52,125,38,126
,125,34,75,125
485 DATA 34,75,250,32,94,250,25,177,125,25,177,125,34,75,3
75,38,126,125
490 DATA 43,52,125,45,198,125,51,97,500,34,75,125,38,126,1
25,43,52,375
495 DATA 45,198,125,38,126,250,34,75,450
999 DATA -1
```

Tips und Tricks für die BASIC-Programmierung

Tips und Tricks für die BASIC-Programmierung Ihres C-64

BASIC-Programme: kürzer und schneller

Der Commodore C-64 Computer ist zwar reichlich mit RAM-Speicher-ausgerüstet. Doch bei großen Programmen empfiehlt es sich im Interesse der Programmablaufgeschwindigkeit und dem verfügbaren Speicherplatz doch einige Vorkehrungen zu treffen.

Hier sollen nun einige Tips zur Verkürzung von BASIC-Programmen gegeben werden. Sie sind aus der Praxis erwachsen, lassen sich aber grundsätzlich auf andere BASIC-Rechner übertragen.

Zunächst seien einige Richtlinien angeführt, die grundsätzlich Vorteile bringen.

1. möglichst viele Befehle in eine Zeile packen
2. unnötige Spaces weglassen
3. REM-Anweisungen weglassen
4. häufig benutzte Konstanten (z. B. 1.02368) einmal in eine Variable speichern, mit dieser arbeiten
5. END-Anweisung in letzter Zeile weglassen
6. immer wieder selbe Variablen für verschiedene Aufgaben benutzen
7. immer wieder gebrauchte Programmteile als Subroutines schreiben
8. bei indizierten Variablen immer auch die nullte mitbenutzen
9. Variablennamen hinter NEXT weglassen

Zusätzlich sind in der täglichen Praxis noch folgende Punkte aufgefallen:

1. Wahre Bytefresser sind die Sprunganweisungen. Zumindest im Commodore-BASIC wird die Zeilennummer Ziffer für Ziffer abgespeichert, so daß ein GOSUB5000 gleich 3 Bytes mehr braucht als GOSUB5 . Wenn man in einem Programm nun einige Subroutines hat, die nicht sehr lang sind und häufig aufgerufen werden, lohnt es sich schon, diese in die einstelligen Zeilennummern zu packen und die seltener benötigten in die zweistelligen. Davor muß zwar ein Befehl stehen, der alle Subroutines überspringt (z. B. 0 DIM ..... :GOTO99), es rentiert sich aber trotzdem.
   Wird in einem ("Menü"-)Programm häufig zu ein und derselben Zeile zurückgesprungen, kann es sich empfehlen, alles als Subroutine zu schreiben: ein RETURN braucht 3 Bytes weniger als ein GOTO100.

2. Viel Speicherplatz belegen auch lange Codetabellen und Texte. Umso schlimmer, wenn man sie dann später gleich doppelt herumstehen hat. Das passiert nämlich immer bei Befehlen wie

   T$="TEILTEXT1TEILTEXT2"

   Der String steht hierbei Zeichen für Zeichen im Programmtext und später auch in der Variablen T$. Das gleiche passiert, wenn man numerische Codetabellen aus DATA-Zeilen liest und in indizierten Variablen speichert.

   Abhilfe bieten hier DATA-Zeilen, die bei jedem Zugriff von vorn nach hinten durchgelesen werden (nach einem RESTORE). Das braucht zwar ein wenig mehr (aber wirklich nur wenig!) Ausführungszeit, kann aber hunderte Bytes Speicherplatz sparen. Um die Laufzeit in Grenzen zu halten, muß man natürlich vorher wissen, was man will, also z. B. die I-te Zahl der Tabelle.
   (Der andere Weg, Codetabellen nämlich nach jedem Programmstart erst einmal von einem Datenfile in indizierte Variablen einzulesen, ist einfach zu mühsam, wenn man keine Floppy hat.)

3. Jeglicher Komfort kostet. Im Extremfall sollte man darüber nachdenken, ob die Variablen wirklich zweibuchstabige Namen haben müssen. Das einfache Alphabet hat immerhin 26 Buchstaben, die für die wichtigsten und häufigsten Variablen ausreichen sollten.
   Auch das Prozentzeichen der Integer-Variablen wird extra abge-

speichert. Diese haben also normalerweise nur Sinn als indizierte Variablen (wegen des dann geringeren Platzbedarfs) mit sehr vielen Elementen, nicht als Einzelvariable !

4. Und wie es sowieso immer ist: vorheriges Überlegen spart am meisten. Also schon bei der Konzeption des Programms festlegen, welche (einfache, einbuchstabige) Variable wofür zuständig ist, in welchen Programmbereichen ihr Wert nicht mehr interessiert und sie also neu und anders verwendet werden kann, ohne daß eine neue Variable nötig wird.
   Oder: nachdem ich mühselig eine 256 Ziffern lange Codetabelle erstellt hatte, die in DATA-Zeilen ins Programm eingebaut war, fiel mir erst auf, daß exakt jede vierte Zahl von vornherein Null war. In DATA-Zeilen konnten so wieder 128 Bytes (64 mals Null und Komma) gespart werden, was die kleine nötige Umrechnung um mehr als 100 Bytes überwog.

5. Man soll eine Programmzeile so voll packen, wie es nur geht. Pro eingesparter Zeilennummer gibt das 4 Bytes. Die Subroutine zur Umwandlung von Dezimal- in Hexazahlen paßt so in eine einzige Zeile, wenn man vorher bezüglich der 16 Ziffernzeichen etwas vorgesorgt hat.

Als zusätzlicher Nutzen ergibt sich, daß alle diese Maßnahmen (außer bei den DATA-Zeilen) das Programm auch schneller machen!

Eins muß natürlich eingestanden werden: kaum eine dieser Maßnahmen erhöht die Übersichtlichkeit und Lesbarkeit eines Programms.
Es empfiehlt sich daher, eine gute Dokumentation des Programmes separat zu verfassen und abzulegen.

Umwandlung Hexadezimal – Dezimal

Zugegeben – Programme zur Umwandlung hexadezimaler Werte in dezimale Zahlen gibt es wie Sand am Meer. Wenn ich diesen hier noch eines hinzufüge, so tue ich das, weil das Programm eine Methode verwendet, die ich noch nirgends beschrieben sah.

Das Umwandlungs-Programm verwendet den ASCII-Code der Hexadezimalziffern zur Berechnung der dezimalen Zahl. Die Länge der umzuwandelnden Hex-Zahl ist nur dadurch begrenzt, daß Strings maximal 255 Zeichen haben dürfen und daß dezimale Werte 1.7E38 nicht überschreiten dürfen. Das Programm ist überdies sehr schnell. Hier die Programmauflistung:

```
10 INPUT"EINGABE HEX-ZAHL";H$:D=0
20 FOR I=1 TO LEN(H$):A=ASC(MID$(H$,I,1))-48:IF A>16 THEN
 A=A-7
25 IF A<0 OR A>15 THEN PRINT"FEHLERHAFTES ZEICHEN BEI DER
 EINGABE":GOTO 20
30 D=D*16+A:NEXT
40 PRINT"DEZIMALER WERT VON $";H$;":";D
```

Falls Sie zusätzlich eine Prüfung auf zulässige Hex-Ziffern einbauen wollen, um etwa Gebilde wie A3KA als ungültig zu erkennen, so fügen Sie bitte die folgende Zeile hinzu:

```
25 IFA<0ORA>15THENPRINT"FEHLERHAFTES ZEICHEN IN
   EINGABE":GOTO20
```

Wenn Sie wollen, können Sie unter Verwendung des ASCII-Codes auch ein DEZIMAL-HEXADEZIMAL-UMWANDLUNGSPROGRAMM schreiben.

INT mit INTEGER

Die BASIC-Anweisung INT ermöglicht bekanntlich eine ganzzahlige Rundung einer Dezimalzahl. Diese Rundung können Sie auch dadurch erreichen, daß Sie die Dezimalzahl einer INTEGER-Variablen (Variablenname und "%"-Zeichen) zuweisen. Beachten Sie jedoch, daß diese Zahl im Bereich von –32768 und +32767 liegt.

Vorteil der Methode der Integer-Zuweisung ist eine geringere Schreibarbeit und eine geringfügig schnellere Ausführungszeit.

Runden von Dezimal-Zahlen

Unter Runden einer Dezimalzahl versteht man das "Abschneiden" dieser Zahl nach einer bestimmten Dezimalstelle, wobei die erste Stelle, die durch das Abschneiden wegfällt, zur Korrektur der davorliegenden Stelle benutzt wird. Ab der Ziffer 5 aufwärts wird die davorliegende Stelle um eins erhöht, im anderen Fall bleibt sie unverändert.

Beispiel:
Rundung auf zwei Stellen nach dem Komma (Geldbeträge)

12.345 liefert nach Rundung 12.35 (die abgeschnittene Ziffer ist 5, daher wird aufgerundet)

12.3449 liefert mit Rundung 12.34 (die abgeschnittene Ziffer ist 4, daher wird die Stelle davor nicht geändert)

Eine Rundung wird vor allem bei Ausgabe von Geldbeträgen benötigt. Das folgende Programm führt diese Rundung durch:

```
100 INPUT"ANZAHL DER DEZIMALSTELLEN";S
110 INPUT"ZU RUNDENDE ZAHL";Z
120 PRINT INT(Z*10↑S+0.5)/10↑S
```

Wenn Sie etwa Geldbeträge runden wollen, so können Sie die folgende Funktion schreiben — diese rundet auf zwei Dezimalstellen:

```
200 DEFFNR(X)=INT(X*100+0.5)/100
210 A=12.345:B=345.6743:C=0.4573
220 PRINT FNR(A);FNR(B);FNR(C);
```

Damit können Sie dann im Print-Befehl diese Funktion aufrufen:

Beispiel:

A=12.345 B=345.6743 C=0.4573

PRINT FNR(A),FNR(B),FNR(C) liefert Ihnen

12.35 345.67 .46

RANDOMIZE-Simulation

Das ansonsten so leistungsfähige und wohldurchdachte Commodore-BASIC wird durch das Fehlen einer RANDOMIZE-Anweisung in seinem Glanz getrübt.

Das wiederholte Aufrufen der Funktion RND(1) liefert Ihnen eine Folge von Pseudo-Zufallsziffern. Schalten Sie das Gerät ein, so bekommen Sie immer die gleiche erste Zufallsziffer und damit, weil jede nachfolgende Zahl aus der vorherigen berechnet wird, immer die gleiche Folge von Zufallszahlen. Dieser Umstand trübt natürlich Spiele, die Zufallsziffern verwenden, da sich immer wieder dieselbe Spielsituation ergeben wird.

Einige BASIC-Versionen besitzen nun den Befehl RANDOMIZE, mit dem die Zufallszahlenfolge initialisiert wird (etwa durch die Zeit).

Es ist jedoch glücklicherweise möglich, diesen Befehl beim Commodore zu simulieren. Der C-64 verwendet die Speicherzellen 139 – 143 (dezimale Adressen) zur Berechnung der nächsten Zufallsziffer. In diese Zellen wird bei RESET oder RESTART (also beim Einschalten des Gerätes) durch die RESTART-Routine ein gewisser Wert geschrieben (Zufallszahlenfolge wird initialisiert).

Sie können diese Zellen natürlich mit POKE-Anweisungen überschreiben und damit einen anderen Startwert für die Zufallszahlenfolge erhalten.

Sinnvoll ist es, diese Initialisierung durch Zeitzähler durchzuführen, da der Startwert der Folge nur vom Zeitraum zwischen Einschalten des Gerätes und Aufrufen des Programmes abhängt. Wenn Sie nun sehr schnell-ändernde Speicherzellen für das Überschreiben der Speicher 139 – 143 verwenden (etwa Mikrosekunden-Zähler), so werden Sie wohl nie dieselbe Zufallszahlenfolge erhalten.

Zwei Zeitzähler sind in Speicher 56324 und 56325 zu finden. Weiterhin können Sie auch die Zellen 160 – 162 verwenden, die alle Zeitzähler sind.

Abschließend gebe ich Ihnen die Möglichkeit an, wie Sie zu Beginn

eines Spielprogrammes die RANDOMIZE-Anweisung ersetzen können:

```
100 POKE 139,PEEK(56324):POKE140,PEEK(56325)
110 POKE 141,PEEK(160):POKE142,PEEK(161)
120 POKE143,PEEK(162)
130 PRINT RND(1)
```

Damit werden Ihre Spiele wirklich "zufällig" und nicht "pseudozufällig".

BASIC-TOKENS und variables Programmieren

BASIC TOKENS und variables Programmieren mit dem Commodore 64

In diesem Teil will ich Ihnen einige außergewöhnliche Anwendungen des vielseitigen BASIC-Befehls "POKE" zeigen.

Zur Wiederholung:

POKE x,y bringt den Wert y in die Speicherzelle mit der dezimalen Adresse x
(0<=x<=65535 und 0<=y<=255)

Dieser Befehl wurde bisher wohl hauptsächlich in Zusammenhang mit der Bildschirmausgabe verwendet. Bei geeigneter Wahl der Werte für x und y kann man den Bildschirmspeicher des C-64 (VIDEO-RAM) ändern und damit effektvolle Zeichnungen auf den Bildschirm "zaubern". Genaueres über POKE und BILDSCHIRM lesen sie bitte im Handbuch.

Mit POKE können Sie aber wesentlich mehr erreichen! Wie Sie Programme schreiben, die sich während der Ausführung selbsttätig ändern, wie Sie damit Speicherplatz und Zeit sparen können und wie Sie Programme schreiben können, die Ihnen das "normale" BASIC nicht ermöglicht, erfahren Sie in diesem Abschnitt.

Grundlagen

Vorerst wollen wir uns mit der Frage beschäftigen, wie im Commodore 64 BASIC-Programme abgespeichert werden.

Wenn Sie BASIC-Anweisungen eintippen, die mit einer Zeilennummer versehen sind, so werden diese Anweisungen nicht wie im "Tischrechner-Modus" sofort ausgeführt, sondern vorerst nur im Programmspeicher abgelegt, wo sie auf die RUN-Anweisung warten.

Dieser Programmspeicher beginnt mit der Speicherzelle 2048 (dezimale Adresse). Jede Programmzeile wird im Programmspeicher in Form eines Blocks abgelegt. Ein solcher Block hat den folgenden Aufbau:

STARTADRESSE DES NÄCHSTEN BLOCKS	ZEILEN-NUMMER	BASIC-ANWEISUNGEN	LEER- oder TRENNBYTE

Nun genauer zu den einzelnen Bestandteilen eines solchen Blocks:

Startadresse des nächsten Blocks

Jeder Block beginnt mit der Startadresse des nachfolgenden Blocks. Die Adresse ist binär codiert in zwei Bytes abgespeichert und gibt an, ab welcher Speicherzelle der nachfolgende Block zu finden ist.

Wie alle Adressen, so ist auch diese Adresse so abgelegt, daß sich das niedrige Adress-Byte in der Speicherzelle mit der niedrigeren Adresse befindet.

Zeilennummer

Die Zeilennummer belegt ebenfalls zwei Bytes des Blocks und ist binär codiert.

BASIC-Anweisungen

In den nachfolgenden Bytes sind die BASIC-Anweisungen abgelegt. Die einzelnen Zeichen (Buchstaben, Ziffern und Sonderzeichen) werden dabei grundsätzlich im entsprechenden ASCII-Code abgelegt und somit belegt ein Zeichen auch im wesentlichen ein Byte des Programmspeichers. Siehe ASCII-Babelle im Commodore Microcomputer Handbuch Seite 135.

Anders ist nun jedoch die Abspeicherung der BASIC-Schlüsselworte (wie etwa RETURN, PRINT usw.). Bei ihrer Abspeicherung geht der

C-64 sehr sparsam um. Jedes Schlüsselwort wird codiert und bekommt einen Wert von 128 bis 255 um nicht mit dem ASCII-Code zu kollidieren. Ein Schlüsselwort belegt daher nur ein (!) Byte des Programmspeichers – egal ob es wie "RETURN" sechs Buchstaben oder wie "IF" zwei Buchstaben hat.

Die Tabelle, die nun folgt, gibt Ihnen die Werte an, durch die die BASIC-Schlüsselworte codiert sind:

BASIC-Schlüsselwort	**CODE**	**BASIC-Schlüsselwort**	**CODE**
END	128	TO	164
FOR	129	FN	165
NEXT	130	THEN	167
DATA	131	STEP	169
INPUT	133	+	170
DIM	134	–	171
READ	135	*	172
LET	136	/	173
GOTO	137	↑	174
IF	139	>	177
RESTORE	140	=	178
GOSUB	141	<	179
RETURN	142	LEN	195
REM	143	STR$	196
STOP	144	VAL	197
ON	145	ASC	198
DEF	150	CHR$	199
PRINT	153	LEFT$	200
CLR	156	RIGHT$	201
OPEN	159	MID$	202
CLOSE	160	π	255
GET	161		

Wie kommt man zu diesen Werten ?

Schreiben Sie dazu bitte das folgende Mini-Programm, um etwa den Code des Schlüsselwortes "INPUT" zu erhalten:

NEW (löscht den Programmspeicher)

1 INPUT

Wenn Sie den Code eines anderen BASIC-Schlüsselwortes haben wollen, so ersetzen Sie bitte INPUT durch dieses.

Stossen Sie sich nicht daran, daß die eingegebene Programmzeile nicht den syntaktischen Regeln von BASIC entspricht – Ihr C-64 tut es nämlich auch nicht – zumindest solange nicht, solange Sie das Programm nicht durch "RUN" starten. Bei Eingabe einer Programmzeile werden vom C-64 keine Syntax-Prüfungen durchgeführt.

Wir haben vorhin erfahren, daß jede Programmzeile in Form eines Blocks in den Programmspeicher gelegt wird – genau dies ist natürlich auch mit der obigen Zeile mit der Zeilennummer 1 geschehen.

Da wir vor Eingabe dieser Zeile NEW getippt hatten, liegt diese Zeile noch dazu am Beginn des Programmspeichers, also ab Speicherzelle 2048.

Um nun die Werte der ersten Programmspeicherzellen zu erhalten, geben Sie bitte die folgende Anweisung ohne Zeilennummer, also im "Tischrechner-Modus" ein:

```
FOR I=2048 TO 2054 : PRINT I,PEEK(I) : NEXT I
```

Sie erhalten damit die folgende Auflistung der Speicherinhalte:

```
2048    0
2049    7
2050    8
2051    1
2052    0
2053    133
2054    0
```

Interpretieren wir nun diese Werte:

In Speicher 2049 und 2050 befindet sich die Startadresse des nachfolgenden Blocks. Beachten Sie bitte die etwas ungewöhnliche Darstellung. Die Adresse ist binär-codiert, das niedrige Adressbyte ist in Speicher 2049, das höhere Adressbyte in Speicher 2050. Durch die PEEK-Anweisung bekommen Sie jedoch den dezimalen Wert des

Speichers. Um eine vollständige Adresse zu bekommen, verfahren Sie bitte wie folgt:

niedriges Adressbyte + höheres Adressbyte * 256

In unserem Beispiel hat das niedrige Adressbyte den dezimalen Wert 7, das höhere Adressbyte den dezimalen Wert 4. Um die vollständige, dezimale Adresse des nächsten Blocks zu erhalten, rechnen wir

8 * 256 + 7 = 2055

In den Speicherzellen 2051 und 2052 finden wir die Zeilennummer. Für diese gilt dasselbe, wie für die Adresse – denn auch die Zeilennummer ist binär codiert und in zwei Bytes abgespeichert. Wir rechnen wie bei der Adresse:

0 * 256 + 1 = 1

Im nachfolgenden Byte (Adresse 2053) finden wir nun den gewünschten Code für das Schlüsselwort INPUT.

Den Inhalt des Speichers 2054 betrachten wir vorerst nicht.

Wenn Sie die Codes anderer Schlüsselworte erhalten wollen, geben Sie nochmals Zeilennummer 1 und das entsprechende Schlüsselwort ein. Sie überschreiben damit die ursprüngliche Zeile mit Nummer 1. Da Sie jetzt bereits wissen, in welchem Speicher der Code des Schlüsselwortes steht, brauchen Sie nur noch

PRINT PEEK (2053)

im Tischrechner-Modus (ohne Zeilennummer) eingeben.

Leer- oder Trennbytes

Jeder Block im Programmspeicher wird durch ein Trennbyte oder Leerbyte abgeschlossen. In diesem letzten Byte eines Blocks befindet sich immer der Wert 0.

Nun verstehen wir auch den Inhalt der Speicherzellen 2054 von oben. Dieses Byte beendet unseren Block, es enthält den Wert 0.

Mit Speicherzelle 2055 beginnt der nächste Block (diese Adresse ist in den beiden ersten Bytes des vorherigen Blocks gespeichert).

Wenn wir folgende Anweisung im Tischrechner-Modus eingeben, so sehen wir ein merkwürdiges Ergebnis:

```
PRINT PEEK(2055), PEEK (2056)          liefert nämlich
   0                    0
```

Da mit Speicher 2055 der nächste Block beginnt, müßten diese beiden Speicherzellen (2055 und 2056) die Startadresse des nachfolgenden Blocks enthalten. Nun hat unser "Mini-Programm" aber nur eine Programmzeile, also gibt es nur einen Block. Damit der C-64 erkennt, daß das Programm nach Block 1 beendet ist, werden durch das Betriebssystem diese beiden Bytes auf Null gesetzt. Damit sehen Sie auch, daß ein END als letzte Anweisung des Programmes entfallen kann, den bei Abarbeitung eines Programmes erkennt der Computer von selbst das Ende durch die beiden Bytes, die auf Null gesetzt sind.

Wir wollen zur Festigung des bisher gesagten ein kleines, konkretes BASIC-Programm betrachten.

Geben Sie bitte folgende Programmzeilen ein, nachdem der Programmspeicher durch NEW gelöscht wurde.

```
10 INPUT A$
20 PRINT A$
30 END
```

Zur Auflistung der Speicherinhalte geben Sie bitte im Tischrechnermodus die folgende Zeile ein:

```
FOR I=2048 TO 2073 : PRINT I,PEEK(I),:NEXT
```

Sie erhalten damit die folgende Auflistung:

2048	0	2062	153
2049	10	2063	32
2050	8	2064	65
2051	10	2065	36
2052	0	2066	0
2053	133	2067	25
2054	32	2068	8
2055	65	2069	30
2056	36	2070	0
2057	0	2071	128
2058	19	2072	0
2059	8	2073	0
2060	20	2074	0
2061	0		

In Speicher 2048 und 2049 finden wir wiederum die Startadresse des nachfolgenden Blocks. Wir rechnen 8*256+10=2058 und wissen damit, daß der nächste Block mit Speicherzelle 2058 beginnt.

Die nächsten beiden Bytes enthalten die Zeilennummer, hier ist sie 0*256+10=10.

Damit ist der "Vorspann" des Blocks beendet. Das nächste Byte (Adresse 2053) enthält den Wert 133. Wir wissen bereits, daß dies der Code für das Schlüsselwort INPUT ist.

Das nächste Byte mit der Adresse 1054 enthält den Wert 32, das ist der ASCII-Code für den Leerraum (Space). Im nächsten Byte haben wir den Wert 65 – den ASCII-Code für das Zeichen "A". Speicher 1032 enthält den Wert 36, was im ASCII-Code dem Dollarzeichen entspricht. Damit haben wir das genaue Abbild unserer Eingabe im Speicher wiedererkannt mit der einzigen Ausnahme, daß im Programmspeicher alle Zeichen codiert sind. Mit Byte 2057 endet der erste Block.

Ab Speicherzelle 2058 finden wir Block 2, der mit Byte 2066 endet. Ab 2067 bis 2072 ist Block 3 abgespeichert, danach zeigen uns die

beiden Bytes 2073 und 2074 das Ende des Programmes im Programmspeicher an, denn diese beiden Bytes sind auf Null gesetzt.

Welche Folgerungen können wir aus den bisherigen Kenntnissen ziehen:

- Mehrfache Anweisungen in einer Programmzeile (durch Doppelpunkte getrennt) spart Speicherplatz, da der "Vorspann" für jede Einzelanweisung 4 Bytes betragen würde.

- Schlüsselworte benötigen nur ein einziges Byte, Zeichen (Buchstaben, Ziffern und Sonderzeichen) brauchen ebenfalls ein Byte und werden im ASCII-Code abgespeichert.
  Die Codes der Schlüsselworte liegen im Bereich von 128 bis 255, der ASCII-Code als 7-Bit-Code im Bereich 0 – 127, so daß keine Verwechslung zu befürchten ist.

Nachdem wir uns eingehend mit den Grundlagen beschäftigt haben, kommen wir nun zum eigentlichen Gegenstand dieses Abschnittes, dem, wie ich es nenne, VARIABLES PROGRAMMIEREN mit dem C-64.

Variables Programmieren

Ich möchte Ihnen hier drei Varianten des variablen Programmierens vorstellen, nämlich

a) VARIABLE SCHLEIFEN

b) VARIABLE VARIABLE

c) EIN BISHER NICHT DENKBARES PROGRAMM

Falls ich Sie nun neugierig gemacht haben sollte, so freut es mich und ich wünsche Ihnen für die folgenden Zeilen viel Vergnügen.

VARIABLE SCHLEIFEN

Das Prinzip, das dem variablen Programmieren zugrunde liegt, sei hier kurz erklärt:

Wir haben gesehen, daß der Commodore 64 BASIC-Schlüsselworte in einem einzigen Byte ablegt. Mit dem Befehl POKE kann man diese Speicherzellen überschreiben und damit während der Programmausführung dieses Programm ändern.

Betrachten wir die beiden Schleifen:

FOR I=A TO B: INPUT A$(I):NEXTI

FOR I=A TO B: PRINT A$(I):NEXTI

Der Schleifenrumpf besteht zum einen aus dem Befehl INPUT, zum anderen aus dem Befehl PRINT.

Durch Verändern des Bytes, in dem das Schlüsselwort INPUT in codierter Form abgespeichert ist, können Sie die Schleife auf die zweite Form bringen.

Betrachten wir das folgende Beispiel:

NEW

10 FORI=1TO10:INPUTA$(I):NEXTI

```
2048  0          2061  58
2049  24         2062  133
2050  8          2063  65
2051  10         2064  36
2052  0          2065  40
2053  129        2066  73
2054  32         2067  41
2055  73         2068  58
2056  178        2069  130
2057  49         2070  73
2058  164        2071  0
2059  49         2072  0
2060  48         2073  0
```

In Speicherzelle 2062 finden wir den Code für INPUT, nämlich den Wert 133. Wenn wir den Befehl

POKE 2062,153

eingeben, dann ändern wir damit den Code auf PRINT. Bitte überzeugen Sie sich davon, daß nun das Programm tatsächlich geändert ist, indem Sie LIST eingeben.

Eine mögliche Anwendung wäre die, daß Sie die obige Schleife als Unterprogramm mit nachfolgendem RETURN abspeichern und vor Aufruf des Unterprogrammes durch POKE die Schleife so "umschalten", wie Sie es benötigen – entweder Eingabe (INPUT, READ) oder Ausgabe (PRINT).

VARIABLE VARIABLE

Sie können natürlich nicht nur das Schlüsselwort einer Anweisung durch POKE überschreiben, sondern auch die dabeistehenden Variablen. Im obigen Beispiel finden wir in Speicher 2063 den Wert 65, was dem ASCII-Code für "A" entspricht. Wollen Sie die Schleife so ändern, daß nicht A$(I), sondern etwa B$(I) einlesen wollen, so geben Sie

POKE 2063,66

ein (66 ist der ASCII-Code für B).

Sie erkennen bereits, wie vielseitig das Prinzip des Überschreibens eines BASIC-Programmes verwendet werden kann.

Zum Abschluß dieses Teiles werde ich nun ein Programm beschreiben, das mit herkömmlichen BASIC-Mitteln nicht möglich wäre, nämlich ein Programm, das Ihnen den Wert bestimmter Variabler ausgibt.

Variabel soll in diesem Programm der Variablenname einer PRINT-Liste sein und nicht wie üblich nur der Wert einer Variablen.

EIN BISHER NICHT DENKBARES PROGRAMM

Geben Sie bitte das folgende Programm ein, nachdem Sie NEW eingegeben haben:

```
10 A=2:B=10:A$="VAR":DD$="STRING"
20 INPUT"WELCHE VARIABLE";H$
30 FORI=2212TO2214:POKEI,32:NEXT
40 FORI=1TOLEN(H$):POKE2211+I,ASC(MID$(H$,I,1)):NEXT
50 PRINT"WERT DER VARIABLEN ";"R";H$;"■:";XX$
60 GOTO 20
```

Bemerkung:
Im Printbefehl bedeutet das "R" das Schalten auf REVERSE-Darstellung, nach H$ wird die INVERSE-Darstellung wieder aufgehoben.

Bitte geben Sie das Programm genau so ein, wie es oben abgedruckt ist, jedes Zeichen und jeder Zwischenraum ist entscheidend, da sich sonst die Adresse für die POKE-Befehle ändern würden.

Der Wert XX$ in der Zeile 50 ist belanglos, da "XX$" ohnehin überschrieben wird.

Was geschieht in diesem Programm ?

In Zeile 20 geben Sie einen String ein (höhchstens 3 Zeichen), der unter H$ abgespeichert wird. In Zeile 30 wird XX$ der Zeile 50 im Programmspeicher durch Leerräume überschrieben (32 ist der ASCII-Code für den "Space").

	2208	146
	2209	58
	2210	34
	2211	59
X	2212	88
X	2213	88
$	2214	36
	2215	0

Danach wird in Zeile 40 der String H$ in diese drei gelöschten Bytes

gebracht. Ist der String H$ nur ein- oder zwei Zeichen lang, so bleiben die restlichen der drei Bytes mit den Adressen 2212 – 2214 auf Space, diese werden bei Abarbeitung des Programmes ignoriert.

Starten Sie nun das Programm und sehen Sie was passiert.

Wir werden aufgefordert, einen Variablennamen einzugeben. Geben Sie etwa "A" ein und sie erhalten umgehend den Wert von A, nämlich 2. Nun wiederholen Sie das Spiel mit den anderen Variablennamen der Zeile 10. Die Zeile 10 ist hier nur deswegen dazugenommen worden, da Sie sonst keine Variablen aufrufen könnten, die Werte verschieden von Null und verschieden vom Leerstring hätten, da durch das RUN die Variablen gelöscht werden.

Wenn Sie dieses kurze Programm am Ende eines Programmes stellen, jetzt natürlich mit anderen Zeilennummern und ohne Zeile 10, wobei Sie dann aber auch die Adressen ändern müssen, so können Sie die Variablen Ihres Programmes sehr angenehm jederzeit aufrufen.

Wie Sie die richtigen Adressen für die POKE-Befehle herausbekommen, müßten Sie nach diesen Ausführungen bereits wissen.

Außerdem möchte ich an dieser Stelle anregen, solche Programme, die Ihnen Variable ausdrucken, noch einmal als Maschinenprogramme zu schreiben und Sie etwa im Kassettenpuffer abzulegen. abzulegen.

Vergessen Sie nicht, wie vielseitig die Möglichkeiten sind, die Sie mit dem variablen Programmieren gegeben sind. Es ist mit diesem Prinzip viel mehr möglich, als hier erwähnt werden konnte.

Warten mit Wait

Warten mit Wait auf dem Commodore 64

Mit POKE und WAIT können bestimmte Pausezeiten ermöglicht werden.

Zum Beispiel:

```
10 POKE162,0:WAIT162,2↑6
20 END
30 POKE162,0:WAIT162,2↑7
40 END
50 POKE161,0:POKE162,0:WAIT161,2↑0
60 END
70 POKE161,0:POKE162,0:WAIT161,2↑1
80 END
90 POKE161,0:POKE162,0:WAIT161,2↑2
100 END
110 POKE160,0:POKE161,0:POKE162,0:WAIT160,2↑0
120 END
```

Die Ermittlung der genauen Zeit ist sehr einfach:

Für WAIT 162 S=S↑X/60
Für WAIT 161 S=256*2↑X/60
Für WAIT 160 S=512*2↑X/60

Die angegebenen Adressen gelten für den Commodore 64.

Programm Schreibmaschine

Mit diesem Programm ist es möglich direkt von der Tastatur in den Drucker zu schreiben. Mit dem Zeichen @ wird auf Breitschrift geschaltet. Jede Druckzeile muß mit RETURN abgeschlossen werden. Die Umschaltung von Groß- auf Kleinbuchstaben erfolgt mit Cursor Up/Down.

```
10 OPEN4,4
20 POKE 198,0:WAIT 198,1:GETA$:PRINTA$;:IFA$="@"THEN40
30 PRINT#4,A$;:GOTO20
40 PRINT#4,CHR$(14);:GOTO20
```

```
DAS IST EIN TEST

DAS  IST  EIN  TEST
```

Warten auf eine Taste

```
100 GET A$:IF A$<>""THEN 100
110 GET A$:IF A$=""THEN 110
120 PRINT"AHA"
```

Bei dieser Darstellung löscht Zeile 100 den Tastaturpuffer. Zeile 110 wartet auf eine beliebige Taste des CBM. Zeile 120 Sprung zu xxx (bzw. RETURN). Eine wesentlich elegantere und kürzere Lösung bieten die BASIC-Befehle POKE und WAIT.

```
100 GET A$:IF A$<>""THEN 100
110 GET A$:IF A$=""THEN 110
120 PRINT"AHA"
200 POKE 198,0:WAIT 198,1:POKE 198,0:GOTO 210
210 PRINT"AHA"
```

Es ist hiermit auch möglich das mehrmalige Drücken beliebiger Tasten

abzuwarten z. B. 2*Taste = WAIT198,2

Bei dieser Lösung ist auch keinerlei Variable erforderlich.

Ein Hinweis sei bei der Verwendung von WAIT angebracht, falsche Adressen nach WAIT veranlassen den C-64 auszusteigen. Zurückholen des C-64 ist nur durch Aus- und Wiedereinschalten möglich.

Testen Sie die Programme mit RUN 100
RUN 110
RUN 200

Einfacher Programmschutz

Diese kleine einfache Routine erlaubt es Ihnen ein BASIC-Programm gegen unbefungten Zugriff zu schützen. Probieren Sie es einfach einmal aus. RUN. Jetzt müssen Sie die Buchstaben TEST hintereinander eingeben und die "Programmtür" wird geöffnet. Codewörter können Sie selbst festlegen.

```
500 PRINT"":Y=0
510 POKE 198,0:WAIT 198,4
520 GET A$,B$,C$,D$
530 IF A$+B$+C$+D$="TEST"GOTO 560
540 Y=Y+1:IFY=2 GOTO 999
550 GOTO 510
560 PRINT"HIER BEGINNT IHR ZU"
570 PRINT"SCHUETZENDES PROGRAMM"
999 END
```

PRINT USING

"PRINT USING" für Commodore 64

Es ist eine bekannte Tatsache, daß der leistungsstarke und kommerziell gut einsetzbare C-64 trotz Verbesserung des Betriebssystems noch immer keine "Print using" Anweisung enthält.

Abhilfe schaffen zwei kleine Unterprogramme, eines für die Formatierung von Texten, ein zweites zum Ausdruck von Zahlen mit übereinanderstehenden Dezimalpunkten.

Der Einsatz dieser einfachen Programm-Module empfiehlt sich dann, wenn in Tabellen Texte und Zahlen in Kolonnen sauber übereinanderstehend gedruckt werden sollen.

Bisher war es so, daß kurze Texte und Zahlen auch entsprechend weniger Zeichen in den Puffer des Druckers geschoben haben, nachfolgende Daten also verschoben waren.

Hier hilft auch nicht die TAB-Anweisung des CBM, da diese nur auf dem Schirm funktioniert, nach außen aber in ein SP (n) umgewandelt wird. Das Prinzip der Unterprogramme (44444 für Text und 55555 für Daten) liegt darin, daß eine maximale Stellenzahl vorgegeben wird und die U-Routinen den Rest mit Leerstellen auffüllen.

Die Vorgaben

Zeile 10:	maximale Anzahl von Vorkommastellen einer Zahl
Zeile 20:	maximale Anzahl von Stellen hinter dem Komma einer Zahl
Zeile 30:	maximale Länge eines Textes

Diese Vorgaben bestimmen das Format des gesamten Ausdrucks. So kann das Fomat auch an bestehende Formulare angepaßt werden.

Im Beispiel steht der Text links, zwei Zahlenkolonnen rechts davon. Texte und Zahlen können beliebig gemischt werden. Die Variablen VK, NK, TL, TT$, TT, ZZ, ZZ$, ZL, ZV, ZI sind belegt und somit für die weitere Verwendung blockiert.

Der Ausdruck einer Zeile erfolgt erst durch die PRINT-Anweisung in Zeile 90. Dieses Prinzip funktioniert natürlich auch bei anderen Rechnern sowie Druckern, bedarf eventuell aber einer Anpassung der Software.

Übrigens:
Die Adresse 44444 und 55555 eignen sich hervorragend für oft anzusprechende Unterprogramme.

Listing mit Beispiel
Wenn zwei Nullen nach dem Dezimalpunkt folgen, werden diese nicht ausgedruckt.

```
5 REM BEISPIEL DRUCKT TEXT UND ZWEI
6 REM ZAHLEN IN EINER ZEILE IM KOLONNEN-DRUCK
7 REM MIT UNTEREINANDERSTEHENDEN KOMMAS
10 VK=9:REM STELLEN VOR DEM KOMMA MAX
20 NK=4:REM STELLEN NACH DEM KOMMA MAX
30 TL=20:REM MAXIMALE TEXTLAENGE
40 INPUT"TEXTEINGABE ";TT$:GOSUB 44444
50 INPUT ZZ:GOSUB 55555
60 INPUT ZZ:GOSUB 55555
90 OPEN4,4,14:PRINT#4:CLOSE4
99 GOTO40
44444 REM UNTERPROGRAMM
44445 TT=LEN(TT$):OPEN4,4,14:PRINT#4,TT$;TAB(TL-TT);:CLOS
E4:RETURN
55555 REM UNTERPROGRAMM FUER ZAHLEN
55558 ZZ$=STR$(ZZ)
55560 ZL=LEN(ZZ$):ZV=0:FORZI=1TOZL:IFMID$(ZZ$,ZI,1)="."GO
TO55580
```

```
55570 ZV=ZV+1:NEXT ZI
55580 OPEN4,4,14:PRINT#4,TAB(VK-ZV);ZZ;TAB((NK+VK)-(VK-ZV
+ZL)+1);:CLOSE4:RETURN
```

```
    ELCOMP                         560.99          23.67
    BITFIRE                        999.67           3.95
    MAIER                            1           6989.55
    MAGICOMP                         3.33       87977.77
```

Ein einfacher Listschutz

Ein einfacher Listschutz

Wer sein BASIC-Programm gegen unbefugte Auslistung schützen will, kann durch einfaches Verstellen der Zeiger sein Programm ''unlistbar'' machen. Wer einen Supermonitor hat, tut sich jetzt beim Experimentieren etwas leichter. Wir wollen zum besseren Verständnis ein kleines Beispiel betrachten.

Starten Sie Ihren Commodore 64 und geben Sie folgendes Programm ein:

```
0 REM
10 FOR Y=1 TO 10
20 PRINT Y
30 NEXT Y
```

BASIC beginnt beim C-64 an der Adresse 800 Hex (2048 Dez). Wie wir bereits im Abschnitt über den Aufbau eines BASIC-Programmes gesehen haben, folgt auf das erste Null Byte (00) sofort ein Zeiger auf die Speicherzelle mit dem nächsten BASIC Befehl. Wir sehen uns die Inhalte der ersten 40 Speicherzellen mit dem Monitor oder mit folgendem Programm einmal an.

```
200 FOR X=2048 TO 2088
210 PRINT PEEK(X);
220 NEXT
```

Adresse	801	802	803	804	805	806	807
0800	00	07	08	00	00	8F	00
Hex		↑ (2050 dez)				(143 dez)	

Wir sehen wie der Zeiger auf die Zelle 807 Hex zeigt. Wir verändern jetzt die 07 in Zelle 802 in 01, so daß der Zeiger auf sich selbst zeigt.

Diese Verbindungselemente (lower Byte / higher Byte) dienen zum verbinden der Programme während List. Bei der Programmausführung werden diese Verbindungselemente nicht benötigt. Um das Programm wieder zurückstellen oder eine Zeile mit niedrigerer Zeilennummer davorzusetzen. Da wir OREM vor das Programm gesetzt haben, ist dies nicht möglich.

Bildschirmausdruck für VC-1515

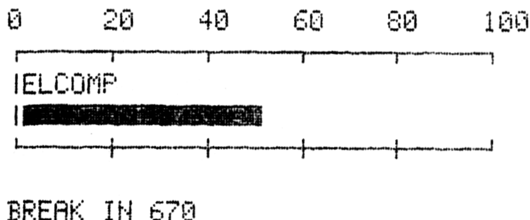
Das folgende Programm stammt aus dem VC-1515 Drucker-Handbuch. Ich habe es auf den C-64 umgeschrieben und getestet. Es arbeitet einwandfrei. Es kann als Unterprogramm an das Programm angehängt werden, dessen Ergebnis auf dem Bildschirm ausgedruckt werden soll.

Ein Aufruf im Direktmodus mit GOSUB 60000 ist auch möglich. Das Programm arbeitet im "Cursor UP" Modus. Wenn Sie ein Programm in "Cursor Down" Modus schreiben, muß G1$=CHR(145) in G1$= CHR(18) in Zeile 60010 ausgetauscht werden.

```
60000 REM BILDSCHIRMAUSDRUCK
60010 G1$=CHR$(145)
60020 OPEN4,4:PRINT#4:G1=1024
60030 FOR G0=0 TO 40:G0$=G1$:G1=G1+40
60040 FOR G2=G1 TO G1+39:G3=PEEK(G2)
60050 IF G3>128THENG3=G3-128:G4=1:G0$=G0$+CHR$(18)
60060 IF (G3>0)*(G3<32)THEN G3=G3+64:GOTO 60100
60070 IF (G3>31)*(G3<64)THEN 60100
60080 IF (G3>63)*(G3<96)THEN G3=G3+128:GOTO 60100
60090 IF (G3>95)*(G3<128)THENG3=G3+64:GOTO 60100
60100 G0$=G0$+CHR$(G3)
60110 IF G4=1THEN G0$=G0$+CHR$(146):G4=0
```

```
60120 NEXT G2:PRINT#4,G0$:NEXT G0
60130 PRINT#4:CLOSE4
60140 RETURN
```

Beispiel eines Bildschirmausdruckes



Stop den Bildschirm

Oft möchte man die Ausgabe von größeren Datenmengen auf dem Bildschirm verlangsamen oder auch anhalten. Hierzu empfiehlt sich folgender Trick. Fügen Sie einfach ein WAIT 198,1 ein.

```
100 DIM B(200)
110 FOR I=1 TO 200 :B(I)=I:NEXT
120 FOR I=1 TO 200 :PRINT B(I):WAIT 198,1:NEXT I
```

Die Zeilen 100 und 110 füllen das Feld BCI mit den Zahlen 1 – 200. In Zeile 120 wird ein Feldelement nach dem anderen ausgedruckt. Hätten wir WAIT 198,1 nicht eingefügt, würden alle 200 Zahlen hintereinander ausgedruckt, ohne daß wir Sie in Ruhe auf dem Bildschirm ansehen können. Der WAIT-Befehl in Zeile 120 stoppt den Befehlsablauf, bis das niederwertigste Bit der Zelle 198 (Bit 0) gesetzt wird. Dies ist der Fall wenn sich darin ein Zeichen befindet. Das Drücken einer zweiten Taste bringt eine zwei, was bedeutet, daß das letzte Bit null wird und Bit 1 = 1 wird.

Geben Sie das oben gezeigte Programm in Ihren C-64 einmal ein und starten Sie es mit RUN. Drücken Sie eine beliebige Taste und dann nach ca. 1 – 2 Sekunden wieder eine Taste. Wiederholen Sie dies öfter.

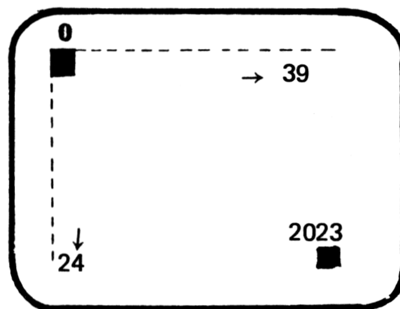
Sie erkennen jetzt den Effekt.

POKE und der Bildschirmspeicher

Der POKE-Befehl bringt hier einen Wert in eine bestimmte Zelle des Bildschirmspeichers. Die Befehlsform sieht wie folgt aus:

POKE A, B
Wobei A = die Adresse in dezimal
B = der Wert in dezimal

Beide Werte A und B können auch Variable sein.
Wir können den POKE-Befehl jetzt dazu verwenden, Zeichen an einer bestimmten Stelle auf dem Bildschirm zu erzeugen. Der Bildschirm kann also wie ein RAM-Speicher betrachtet werden.



Der erste Punkt oben links ist die Adresse 1024.
Beispiel: POKE 1024,1
POKE 1024,49

"49" = Zahl die 1 entspricht.

B darf nur zwischen 0 und 255 liegen (siehe Tabelle).

Will man ein Zeichen an einer anderen Stelle darstellen, muß immer 1 dazugezählt werden. Mit 40 kommen Sie in die nächste Zeile.

Unterprogramm: Zeichen durch C gegeben
L = Zeilennummmer (1 – 25)
P = Position (1 – 40)

```
1000 POKE 1024 + (40*(L–1))+(P–1),C
1010 RETURN
```

Bitte beachten Sie, daß der C-64 bei den oben vorher beschriebenen POKE-Befehlen die Zeichen immer in der jeweiligen Farbe des Cursors auf dem gerade gewählten Hintergrund darstellt. Wollen Sie die Farbe ändern, müßen Sie den Wert für die gewünschte Farbe noch in die entsprechende Zelle des Farbspeichers schreiben.

Der Farbspeicher hat im Normalmodus die gleiche Anzahl von Speicherzellen wie der Bildschirmspeicher und beginnt bei Adresse 55296 dezimal.

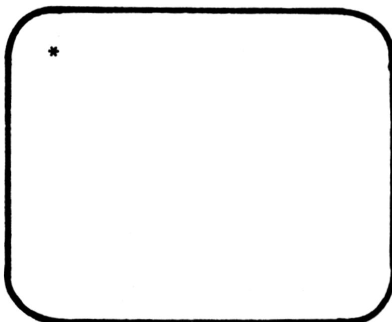
PEEKing in den Bildschirm

PEEK (L)	L = Platz in dezimal im Speicher
C = PEEK (L)	Nimmt den Inhalt von L und macht es zur Variablen C. Es wird immer der Code entsprechend der Tabelle gezeigt.

Home * Return

PRINT PEEK 1024
42
42 = * Asterik-Sternchen

POKE 1024,42



Auf diese Weise können Sie die Codewörter für jedes Zeichen herausfinden.

Zeichencodierung

Zeichen können auf dem Bildschirm auf drei verschiedene Arten dargestellt werden.

1. Durch Poken des zugehörigen Zeichens in den Bildspeicher.
   Bildspeicheradresse von 1024 bis 2023 Dez.
2. Ausdrucken durch PRINT
3. PRINT CHR$(X)

Zwei verschiedene umschaltbare Zeichensätze sind im C-64 vorhanden.

1. Nach dem Einschalten muß Zelle 53272 21 dezimal enthalten.
2. Im Betrieb für Kleinbuchstaben wird in die Zelle 53272,2 eingepoked.

Beispiel:
Die Zahl 83 in Zelle 1024 ergibt ein Herz. Wird die Zahl 211 eingepoked, wird das Herz negativ dargestellt. Dem Buchstaben S entspricht als Wert die Zahl 19. Negativ wird S durch die Zahl 147.

Beispiel:

```
10 POKE 1024,83
RUN
```

Das Ausdrucken durch PRINT geschieht über den PRINT CHR$-Befehl. Hierzu finden Sie den Wert ganz rechts in der Spalte (Tabelle Anhang F im mitgelieferten Handbuch).

POKE 1024,83 und PRINT CHR$(211) bewirken das gleiche. Mit beiden Befehlen kann vom Programm her eine Kontrolle über die einzelnen Zeichen erfolgen. Bei PRINT CHR$ erfolgt die Ausgabe jedoch nicht an einer bestimmten Stelle, sondern an der momentanen Cursorposition.

Da in vielen Listings die Grafikzeichen für die Bildschirmsteuerung nicht immer genau zu erkennen ist, haben wir noch einmal für Sie alle Cursor-Steuerzeichen zusammengestellt.

Da es nicht immer einfach ist, in einem BASIC-Listing die Graphiksymbole zu erkennen, hier eine kurze Zusammenstellung.

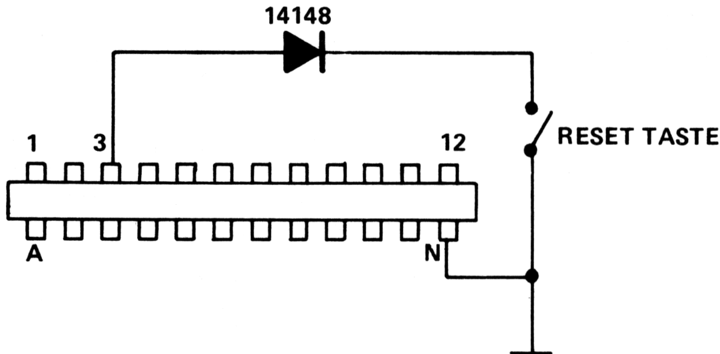
ASCII	PRINTS	DESCRIPTION	
146		Reverse Aus	Use CTRL key
18	R	Reverse Ein	Use CTRL key
3	C	RUN / STOP	
147		Bildschirm löschen	
19	S	Cursor nach oben links	
145		Cursor nach oben	
17	Q	Cursor nach unten	
157		Cursor nach links	
29		Cursor nach rechts	
148		Insert	

Einige wichtige Adressen und deren Funktionen

Hex	Dez	
FF9F	65439	Tastaturabfrage
FFA5	65445	Holt ein Byte vom IEC-Bus
FFA8	65448	Sendet ein Byte über IEC-Bus
FFC0	65472	Eine logische Datei wird geöffnet (entspricht OPEN) log. File # im Akku.
FFC3	65475	Eine logische Datei wird geschlossen (entspricht CLOSE) log. File # im Akku.
FFCC	65484	CLRCH schaltet die Ein/Ausgabe zurück auf Bildschirm und Tastatur.
FFCF	65487	Holt ein Zeichen aus dem Akku.
FFD2	65490	Gibt das momentane Zeichen im Akku aus (CBM ASCII). LDA #$93 löscht Bildschirm JSR FFD1 löscht Bildschirm
FFD5	65493	Lädt ein Programm in den Speicher.
FFD8	65496	Speichert ein Programm auf Cassette oder Diskette (je nach dem welcher logische File gerade aktiviert ist). DEFAULT (0) ist Cassette.
FFE4	65508	GET Funktion. Ein einzelnes Zeichen wird in den Akku gebracht.
FFE7	65511	Alle Ein- und Ausgabekanäle werden zurückgesetzt (auf DEFAULT).
F88D	63629	Cassettenmotor einschalten
F8D6	63702	Cassettenmotor
FC9D	64669	ausschalten

RESET Knopf für Commodore 64

Commodore did it again! Sie haben wieder einmal einen Computer gebaut, der keine RESET-Taste hat. Der einfachste Weg einen eigenen RESET-Knopf noch nachträglich zu installieren ist, daß am User Port der RESET Anschluß über einen Schalter an Masse gelegt wird.



Der Schalter sollte irgendwo hinten am Gehäuse angebracht werden. Wenn Sie einen Stecker auf den User Port stecken und den Schalter außerhalb vom Gehäuse lassen, ist das Projekt in ca. 5 Minuten durchgeführt. Die Diode können Sie am Schalter anlöten. Bitte achten Sie bei eigenen Hardware Experimenten darauf, daß Sie Ihre Garantieansprüche nicht verlieren.

Abspeichern von Variablen auf Diskette

Mit Hilfe des Befehles "SAVE" können Sie in BASIC den Speicherbereich auf Cassette oder Diskette ablegen, der durch folgende Zeiger in der ZERO-Page (0000 – 00FF Hex) festgelegt ist:

1. RAM Anfang:
   Anfangsadresse niederwertiges Byte in 2B hex = 43 dez
   Anfangsadresse höherwertiges Byte in 2C hex = 44 dez
2. Variablen Anfang
   Anfangsadresse niederwertiges Byte in 2D hex = 45 dez
   Anfangsadresse höherwertiges Byte in 2E hex = 46 dez

3. Variablen Ende = Array Anfang
   Anfangsadresse niederwertiges Byte in 2F hex = 47 dez
   Anfangsadresse höherwertiges Byte in 30 hex = 48 dez

Die Zeiger in den Zellen 37 und 38 hex (55 und 56 dez) markieren das Ende des BASIC-Bereiches.
Wenn wir nun die Zeiger so vertauschen, daß die Zeiger für Anfang und Ende des "normalen" BASIC Programmes jetzt auf den Anfang und das Ende der Variablenliste zeigen, können wir die Variablen auf Diskette abspeichern und wieder einlesen. Von einem nachfolgenden Programm können die Variablen dann wieder eingelesen und übernommen werden. Wenn Sie ab der Variablen Liste bis zum Ende des verfügbaren BASIC RAMs abspeichern, übernehmen Sie auch die Strings und Felder. Wer diese nicht braucht, speichert einfach nur vom Anfang der Variablen bis zum Ende der Variablen ab.

Für unserer Testzwecke grenzen wir den Speicherbereich des C-64 etwas ein, damit die Bereiche, die wir abspeichern wollen, nicht zu groß werden.

```
5 REM VARIABLEN SAVE DEMO
10 POKE 56,35
20 A=1:B=2:C=3:D=4:E=5
30 A$="ELCOMP"+"":B$="BITFIRE"+""
40 FOR X=0 TO 15
50 POKE 49152+X,PEEK(43+X)
60 NEXT X
70 POKE 43,PEEK(45):POKE44,PEEK(46)
80 POKE 45,PEEK(55):POKE46,PEEK(56)
90 SAVE"@0:FILE",8,1
100 POKE45,PEEK(43):POKE46,PEEK(44)
110 POKE43,1:POKE44,8
120 END
200 IF PEEK(1024)=255 THEN 270
210 POKE 1024,25
220 POKE 56,35
230 FOR X=0 TO 15
240 POKE43+X,PEEK(49152+X)
250 NEXT X
260 LOAD"0:FILE",8,1
270 PRINT A,B,C,D,E
280 PRINT A$,B$
290 END
```

Das Demo-Programm erlaubt uns mit RUN einige Variable auf Disk zu speichern und mit RUN 200 wieder einzulesen. Nach Zeile 260 hält das Programm an und es muß GOTO 270 über die Tastatur eingegeben werden, um zu prüfen ob die Variablen auch sicher da sind.

Dieses Programm soll ein Denkanstoß für Ihre eigenen Experimente sein. Der Speicherbereich wurde in Zeile 10 und 220 begrenzt, damit das Test-File auf Diskette nicht zu groß wird.

Verbesserte Version

Eine geringfügig veränderte Version des vorher gezeigten Programmes erlaubt uns, daß wir nach dem Laden der Variablen unser Programm automatisch weiter arbeiten lassen. Bei Verwendung des LOAD-Befehls in Zeile 260 liefert das Programm nach Ausführung dieser Zeile ein READY und stoppt. Um dies zu verhindern, arbeiten wir jetzt anstelle des LOAD-Befehles mit dem SYS-Kommando. Wir rufen die BASIC-Routine für LOAD an der Adresse F4AB hex = 62635 dez auf. Vorher müssen wir jedoch einige Zeiger in der Zeropage selbst setzen und vorbereiten.

Den File des Namens packen wir in Zeile 251 in den geschützten BASIC-Bereich. Die dezimalen ASCII-Werte für den Filenamen FILE poken wir in den Adressbereich ab 49407.

F = 70
I = 73
L = 76
E = 69

Den Zeiger, der auf diese Adresse mit dem Namen zeigt setzen wir in den Zellen 187 und 188.
Adresse = 49407 = C0FF Hex. C0 = 192, FF = 255
Also POKE 187,255 und POKE 188,192.

In Zeile 255 legen wir in der Zeropage-Adresse 183 die Länge des Filenamens mit 4 fest (FILE = 4 Buchstaben). Mit POKE 147,0 wird auf LOAD geschaltet. Eine "1" in dieser Zelle bedeutet VERIFY. In Zeile 257 wird der Kanal und in Zeile 258 die Sekundäradresse bestimmt.

Sie können diese Technik auch verwenden, um Programme mit einer Autorunfunktion auszustatten oder Programme zu entwerfen, die sich selbst auf Diskette ablegen und wieder aufrufen.

```
5 REM VARIABLEN SAVE DEMO
10 POKE 56,35
20 A=1:B=2:C=3:D=4:E=5
30 A$="ELCOMP"+"":B$="BITFIRE"+""
40 FOR X=0 TO 15
50 POKE 49152+X,PEEK(43+X)
60 NEXT X
70 POKE 43,PEEK(45):POKE44,PEEK(46)
80 POKE 45,PEEK(55):POKE46,PEEK(56)
90 SAVE"@0:FILE",8,1
100 POKE45,PEEK(43):POKE46,PEEK(44)
110 POKE43,1:POKE44,8
120 END
200 IF PEEK(1024)=255 THEN 270
210 POKE 1024,25
220 POKE 56,35
230 FOR X=0 TO 15
240 POKE43+X,PEEK(49152+X)
250 NEXT X
251 J=49407:POKEJ,70:POKEJ+1,73:POKEJ+2,76
252 POKE J+3,69:POKE187,255:POKE188,192
255 POKE 183,4 :POKE 147,0
257 POKE 186,8
258 POKE 185,1
259 SYS(62635)
260 RESTORE
270 PRINT B,C,D,E,A
280 PRINT A$,B$
290 END
```

Komfortable Bildschirmpositionierung

Oft ist es elegant, wenn man möglichst viele PEEK und POKE Befehle in BASIC verwendet. Es zeigt dem Leser oder späteren Anwender des Programmes, daß man recht tief in die Maschine eingestiegen ist. Für die Transportabilität des Programmes von einem PET auf einen VC-20 oder auf einen C-64 treten dann jedoch Probleme auf. Es ist auf jeden Fall zweckmäßiger, wenn man sein Programm so auslegt, daß es möglichst transportabel ist.

Beispiel: Positionierung des Cursors auf dem Bildschirm:

Starten Sie die beiden Programme mit RUN und RUN 30. Löschen Sie vorher den Bildschirm. Beide Programme erledigen die gleiche Aufgabe. Mit dem Programm RUN 30 sind sie jedoch transportabler geblieben. In Zeile 30 bedeuten die Grafikzeichen Cursor Home und 25 x Cursor nach unten).

```
10 POKE 214,12:PRINT:POKE 211,19:PRINT"HIER"
20 END
30 PP$="[illegible]"
40 PRINT LEFT$(PP$,11):PRINT TAB(19)"HIER"
50 END
```

Abfrage von Anwender-Eingaben

Dieses Problem stellt sich dem Programmierer fast täglich. Viele Programme fragen den Anwender: "Antworten Sie mit Ja oder Nein, geben Sie 0 für Nein oder 1 für Ja ein". Oder das Programm fragt "JA/ NEIN" usw. Um den Anwender so weit wie möglich zu entlasten, sollte man seine Antwort so einfach wie möglich gestalten.

10 PRINT: "BENOETIGEN SIE ANWEISUNGEN ?"

```
100 PRINT"BRAUCHEN SIE ANWEISUNGEN? "
110 GET A$:IF A$=""THEN 110
120 IF A$="J" THEN GOSUB 3000
130 REM PROGRAMM
140 FOR T=1TO3000:NEXT T
150 PRINT"NOCHEINMAL"
160 GET B$:IF B$=""THEN 160
170 IF B$="J"THEN 130
```

```
180 END
3000 PRINT"HIER DIE ANWEISUNGEN"
3010 RETURN
```

Mit dieser einfachen Routine können Sie abfragen, ob man Anweisungen benötigt, und ob man das Programm noch einmal wiederholen möchte. Als Programmersatz wurde die Zeile 140 eingefügt. Hier sollte der Anfang Ihres Programmes sein. Ab Zeile 3000 sollten die Anweisungen stehen.

Druckerformattierung

Wenn Sie mit Hilfe des TAB-Befehles eine Ausgabe auf den Drucker geben wollen, sieht der C-64 in der momentanen Cursorposition (Adresse 211 dezimal) nach. Wenn die Zahl im TAB-Befehl kleiner als die Cursorpositon ist, dann wird der TAB-Wert einfach ignoriert und an die momentane Position gedruckt. Wenn der TAB-Wert größer ist, wird die Differenz gebildet und diese in entsprechende Cursorbewegungen nach rechts umgesetzt.

Wenn man auf einen Drucker ausgibt, ist der Cursor meist in Position 0 und TAB arbeitet, dann wie die SPACE-Funktion SPC. Um TAB richtig auf dem Drucker anzuwenden, kann man die Daten zuerst auf dem Bildschirm und dann auf den Drucker ausgeben. Das nachfolgende Hilfsprogramm zeigt Ihnen einen dynamischen Print-Befehl. Der Text im String wird praktisch gleichzeitig auf dem Bildschirm und Drucker ausgegeben.

```
200 REM AUSGABE AUF BILDSCHIRM UND DRUCKER
205 X=0
210 OPEN3,3,1
220 OPEN 4,4
230 PRINT#3+X,"ELCOMPHOFACKERBITFIRE";
235 IF X THEN PRINT#4,CHR$(13);
240 X=1-X:IFXTHEN 230
245 CLOSE3:CLOSE4
250 END
```

Die dynamische PRINT-Anweisung eignet sich bestens, wenn die Daten,

welche auf dem Drucker ausgegeben werden sollen, sich in Anführungszeichen befinden. Wenn Variable verwendet werden, ist es einfacher die Ausgabebefehle jedes mal neu zu schreiben.

Ausgabeformattierung

Die nachfolgende Routine erweist Ihnen gute Dienste, wenn Sie Zahlen aufgerundet und mit Konstant zwei Stellen hinter dem Komma ausgeben wollen. Auch kleine Zahlen werden als"vernünftige" Beträge ausgegeben. Normalerweise erhalten Sie auf PRINT 0.009 9E – 03. Unser Programm liefert Ihnen hier aufgerundet 0.01 wie Sie dies auch in Ihren Programmen benötigen. Sie können die Ausgaben auch auf einen Drucker leiten. Auch negative Vorzeichen werden an die erste Stelle vor der ersten Zahl gesetzt, egal wie lange die Zahl ist.

Bitte achten Sie bei der Eingabe genau auf die Werte. Die Routine arbeitet einwandfrei. Als Dezimalpunkt muß der Punkt eingegeben werden.

```
100 Q$=".00":LQ=LEN(Q$):LT=LOG(10)
110 P=LQ-1:IF Q$=""THEN P=0
115 PRINT"❐":PRINT:PRINT:PRINT
150 INPUT"EINGABE NUMMER";A
160 A1=INT(A*10↑P+0.5)/10↑P
165 IF A1=0 THEN 220
170 B1$=STR$(A1)
180 B2$=STR$(A1*10↑P)
190 LG=INT(LOG(ABS(A1))/LT)
195 IF ABS(A1)=1THEN B$="1"+Q$
200 IF ABS(A1)<1 THEN B$=LEFT$(Q$,ABS(LG))+RIGHT$(B2$,LEN
    (B2$)-1)
210 IF ABS(A1)>1 THEN B$=MID$(B1$,2,LG+1)+","+RIGHT$(B2$,P)
215 IF ABS(A)=1 THEN B$="1"+QS
220 IF A1=A THEN B$=Q$
225 IF A1<0 THEN B$="-"+B$
230 PRINT"❐"
240 PRINTA1,B$
245 PRINT:PRINT
250 GOTO 150
```

Die Variable A1 verwenden Sie zum Rechnen. B$ dient zur Darstellung.

Die nachfolgende Änderung bringt die Ausgabe auf einen Drucker. Achtung: Die Dezimalpunkte werden hier nicht untereinander gedruckt. Hierzu bedarf es einer besonderen Formattierung.

```
225 IF A1<0 THEN B$="-"+B$
230 PRINT"]"
235 OPEN4,4
240 PRINT#4,A1,B$
242 CLOSE4
245 PRINT:PRINT
250 GOTO 150
```

Wenn Sie sich den String B$ auf drei Stellen hinter dem Komma ansehen wollen, ändern Sie in Zeile 100 Q$ = ".000" ab.

```
100 Q$=".000":LQ=LEN(Q$):LT=LOG(10)
```

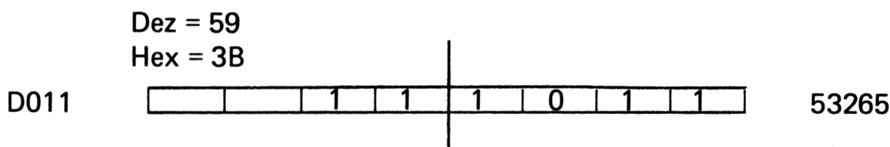
NOTIZEN

Hochauflösende Grafik mit dem C-64

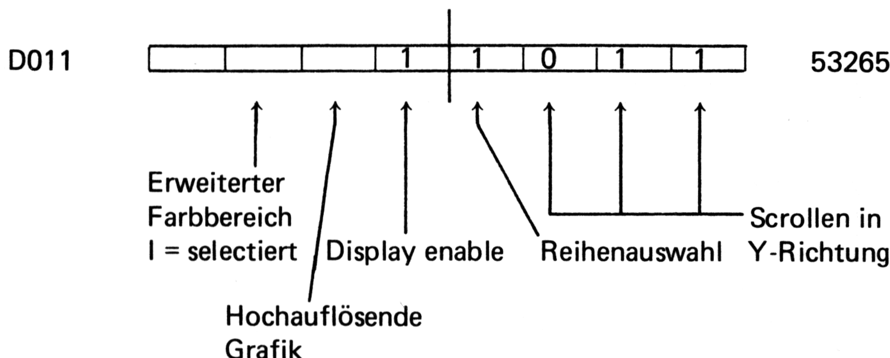
Nachdem wir uns eingehend mit dem Zeichensatz und den Möglichkeiten der Bildschirmspeicher-Konfiguration beschäftigt haben, sind wir in der Lage uns der hochauflösenden Grafik im C-64 zuzuwenden.

Unter hochauflösender Grafik verstehen wir hier die Möglichkeit, jeden einzelnen Bildpunkt getrennt anzusteuern (320 x 200 Punkte). Im Zeichensatzmodus konnten wir immer nur Zeichen, bestehend aus einer 8 x 8 Matrix von Punkten, gleichzeitig ansprechen.

Die hochauflösende Grafik wird mit POKE 53265,59 eingeschaltet.



Nach dem Einschalten enthält die Registeradresse 53265 Ihres C-64 den Wert 27 dezimal.

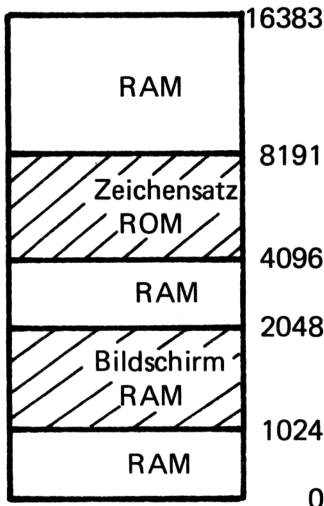


Wir erkennen aus dem Register wie man mit POKE 53265,59 die hochauflösende Grafik einschalten (Bit wird gesetzt) und mit POKE 53265,27 wieder abschalten kann. Das Initialisieren der hochauflösenden Grafik bei Betrieb in Block 1 verändert die Aufteilung die Blöcke insofern, daß jetzt der Farbspeicher über die vier höherwertigen Bit der Zelle 53272 festgelegt wird. Der Zeichensatz bleibt an seiner Stelle und die Lage des neuen 8k großen Bildschirmspeichers wird durch die drei Bit festgelegt, die früher die Lage des Zeichensatzes bestimmt haben.

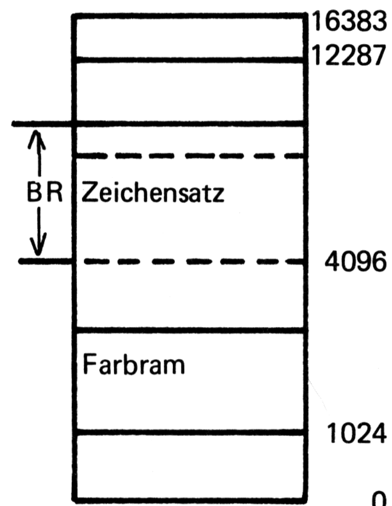
Sie können dies leicht nachprüfen, indem Sie

	POKE 53265,59	und darauf folgend dann
blind eingeben:	PRINT PEEK(53272)	
	POKE 53265,27	

Die Schrift erscheint als Farbe, da die Zeichen als Farben interpretiert werden. Jedes Zeichen bestimmt mit dem Wert seiner Matrix einen bestimmten Farbwert.
Sie werden nach dem ersten POKE feststellen, daß wirre Zeichen den Bildschirm füllen und sich auch nicht bewegen lassen. Um dies zu verstehen, müssen wir wieder den Block 1 heranziehen.



Block 1
nach dem Einschalten



Block 1
nach POKE 53265,59

Die Anfangsadresse des Bildschirm RAMs liegt normalerweise bei 1024 Dez. Durch Einschalten des hochauflösenden Modus, wird der Bildschirmspeicher von 1k RAM auf 8k RAM vergrößert. Unser Zeichensatz ist jetzt durch das Bildschirmfenster sichtbar, da er vom Bildschirm RAM überlagert wird.

Der Teil ohne Grafikzeichen auf dem Bildschirm ist der freie RAM-Teil, der noch oberhalb unseres vorherigen Bildschirmspeichers lag. Man könnte jetzt Ordnung schaffen, indem man das Register des VIC-Bausteines entsprechend setzt und den Bildschirmspeicher und Farbspeicher löscht.

$D018	0	0	0	1	1	0	0	1/0	53272 Dec

Inhalt Hex = 18 oder 19
Dez = 24 oder 25

Den Bildschirmanfang legen wir auf die Anfangsadresse 8192 Dez = 2000 Hex. Den Zeichensatz müssen wir an Adresse 4096 belassen.

Das auf 8k vergrößerte Bildschirmram legen wir nach Adresse 8192 dezimal. Der Farbspeicher wird mit den vier höherwertigen Bit auf 1024 Anfangsadresse gelegt. Wir erreichen dies durch

POKE 53272,24

Jetzt sind unsere Speichergrenzen in Block 1 festgelegt. Wir haben nun einen 8k Bildschirmspeicher ab Adresse 8192. Dieser beinhaltet nun 40 x 20 Zeichenstellen a' 8 Byte. Jedes dieser 8 Byte belegt die Fläche eines normalen Zeichens und repräsentiert 64 Bit wobei jedes Bit einen einzelnen Bildpunkt festlegt. Jetzt wird jeder Punkt durch ein Bit repräsentiert. Im normalen Zeichensatz Modus kann man an eine Speicherstelle, die durch ein Byte adressiert wird, nur ein Zeichen, bestehend aus 8 Byte (8 x 8 Matrix), gleichzeitig bringen.

Beispiel POKE 1064,65

bringt ein A in die erste Zelle in der zweiten Zeile. Es ist besonders wichtig, daß man sich diesen Unterschied zwischen normalem Zeichensatzbetrieb und hochauflösender Grafik vor Augen hält.

Führen Sie am besten selbst noch einige Versuche durch und schalten Sie den C-64 einfach mit POKE 53265,59 in den Grafikmodus um. Ein vorher eingegebenes Listing erscheint jetzt als eine Reihe farbiger Kästchen. Dadurch, daß nach dem Umschalten auf Grafik, das vorherige Bildschirm RAM zum Farbspeicher wird, werden die Zeichen nicht mehr als Buchstaben und Zahlen, sondern als Farbwerte interpretiert.

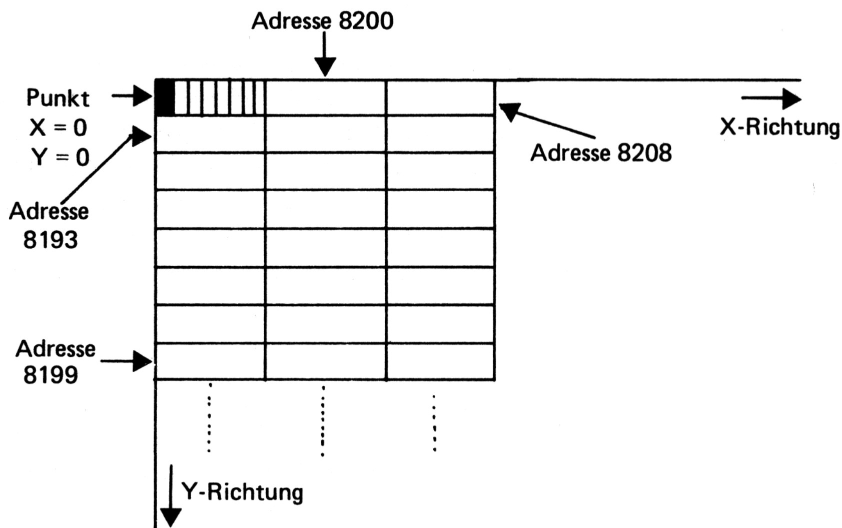
Geben Sie anschließend POKE 52272,24 blind ein, es verschwindet der Hintergrund mit den Grafikzeichen, da der neue Bildschirmspeicher jetzt bei Adresse 8192 beginnt.

Zurück zur gewohnten Darstellung finden Sie, in dem Sie "blind", also ohne Bildschirmanzeige

```
POKE 53272,21
POKE 53265,27
```

eingeben.

Da der Commodore 64 in seiner Grundausstattung, wie sie Mitte 1983 zur Auslieferung kam, keine speziellen BASIC-Befehle für den Bit MAP Modus hat, wollen wir uns eine kleine Routine selbst erstellen. Die Anfangsadresse des Bildschirmes ist 8192 dez. Die Zählung der einzelnen Bit (Bildpunkte) erfolgt so, daß der Punkt in der äußersten oberen linken Ecke die Koordinaten X = 0 und Y = 0 hat. Dies bedeutet, daß die Zählung mit dem ersten Bit der ersten Bildschirmadresse 8192 links oben beginnt.



Die Lage der einzelnen Punkte ist durch zwei Merkmale gekennzeichnet.

1. Durch die Bildschirmadresse in der sie liegen (Byte)
2. Durch das Bit innerhalb dieses Bytes.

Das nachfolgende Beispielprogramm verdeutlicht Ihnen am besten, wie die einzelnen Bildpunkte hintereinander auf dem Bildschirm liegen, und wie die einzelnen Adressen bei einer Bewegung in X-Richtung durchlaufen werden.

```
120 POKE 53265,59
130 POKE 53272,24
140 FOR V=1024 TO 2023
150 POKE V,16
160 NEXT V
170 FOR T=8192 TO 16383
180 POKE T,0
190 NEXT T
192 X=0:Y=0
194 X=X+1
200 NP=Y*320+X
210 AA=8192
220 Q=INT(NP/8)
230 R=(NP/8-Q)*8
270 M=2↑(7-R)
290 I=PEEK(Q+AA)
300 POKE Q+AA,I OR M
310 GOTO 194
```

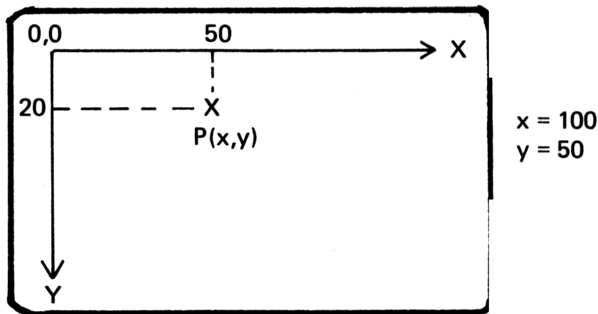
Entwurf eines kleinen Programmes zur Erstellung hochauflösender Grafik

Routine zum Zeichnen eines Punktes

Wir wollen ein kleines Programm entwerfen, welches uns erlaubt, an einer beliebigen Stelle auf dem Bildschirm einen Punkt zu zeichnen.

Unser Koordinatensystem legen wir mit dem Bildschirmanfang zusammen. Dies ist einfacher, als wenn wir die herkömmliche Darstellung

wählen würden, bei der Nullpunkt X = 0, Y = 0 in der linken unteren Ecke liegt.



Zuerst müssen die Werte für X und Y eingegeben werden. Dann wird auf hochauflösende Grafik geschaltet und die Speicheraufteilung in Block 1 selektiert. Dann löschen wir den Farbspeicher und den Bildschirmspeicher. Nun wird die Lage des Punktes berechnet. YP und XP sind die ganzzahligen Byte Werte für die Koordinaten Y und X. A1 ist die Adresse des ersten Bytes der Druckposition. AY ist die Adresse des Bytes in welcher der Punkt gezeichnet werden soll. AA ist die Bildschirmanfangsadresse. R ist der Rest nach der Byteadresse. M ist der Punkt innerhalb des Bytes, welcher gedruckt werden soll. Der Inhalt der Adresse in der das Bit gesetzt werden soll, wird mit PEEK (AY + AA) ausgelesen und mit M geodert, da die Wertigkeit des Bytes von rechts nach links ansteigt, die Punkte aber von links nach rechts gezählt werden.

```
5 INPUT"X-WERT ";X
7 INPUT"Y-WERT ";Y
10 POKE 53265,59
20 POKE 53272,24
30 FOR V=1024 TO 2023
40 POKE V,16
50 NEXT V
60 FOR T=8192 TO 16383
70 POKE T,0
80 NEXT T
115 YP=INT(Y/8)
120 XP=INT(X/8)
```

```
130 A1=(YP*40+XP)*8
140 AY=Y-8*YP+A1
150 AA=8192
160 R=X-8*XP
170 M=2↑(7-R)
180 I=PEEK(AY+AA)
190 POKE AY+AA,I OR M
```

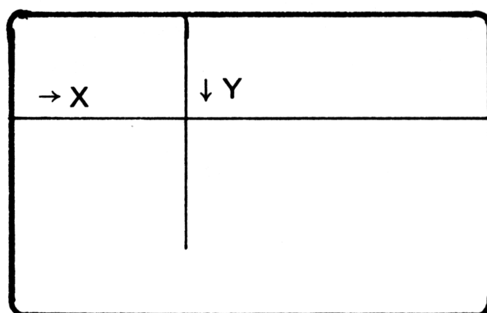
Geben Sie das vorher gezeigte Programm ein und testen Sie es mit den Werten X = 100 und Y = 50. Nach einer Weile muß an der Stelle X = 100, Y = 50 ein winziger Punkt erscheinen. Wir sind also jetzt in der Lage, innerhalb der Grenzen X = 320 und Y = 200 an jede beliebige Stelle einen Punkt zu zeichnen. Funktionen, die in der Form x = f (y) vorliegen, können bereits dargestellt werden, indem zu jedem Wertpaar der entsprechende Punkt ermittelt und auf dem Bildschirm dargestellt wird.

Als nächstes wollen wir gerade Linien in X-Richtung und Y-Richtung ziehen. Hierfür verwenden wir das folgende Beispielprogramm. Als Unterroutine verwenden wir wieder unser bereits bekanntes Punkt-Zeichenprogramm.

```
10 REM HIGRA7 DEMO
42 POKE 53265,59
44 POKE 53272,25
45 FOR V=1024 TO 2023
46 POKE V,16
47 NEXT V
48 FOR T=8192 TO 16383
49 POKE T,0
50 NEXT T
52 GOTO 1000
200 YP=INT(Y/8)
210 XP=INT(X/8)
220 A1=(YP*40+XP)*8
230 AY=Y-8*YP+A1
240 AA=8192
250 R=X-8*XP
260 M=2↑(7-R)
270 I=PEEK(AY+AA)
```

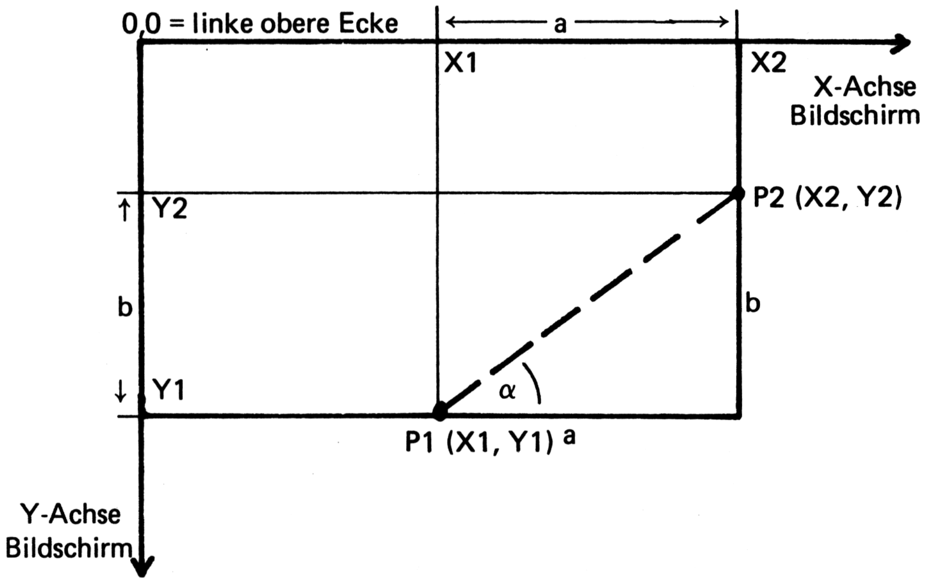
```
280 POKE AY+AA,I OR M
290 RETURN
1000 Y=50
1010 FOR X=0 TO 319
1020 GOSUB 200
1030 NEXT X
1040 X=100
1050 FOR Y=0 TO 159
1060 GOSUB 200
1070 NEXT Y
1075 FOR T= 1 TO 5000:NEXT T
1080 POKE 53265,27
1090 POKE 53272,21
1100 END
```

Dieses Programm zeichnet zwei gerade Linien auf dem Bildschirm. Die X-Linie ist etwas heller als die Linie in Y-Richtung.



Das nachfolgende Programm ist der nächste Schritt auf unserer Expedition in das Reich der hochauflösenden Grafik. Wir wollen nun eine Gerade von einem Punkt zu einem anderen Punkt auf dem Bildschirm zeichnen.

Dazu stellen wir folgende Überlegung an (siehe nächste Seite):



$$\tan \alpha = \frac{b}{a} = \text{Steigung L}$$

$$b = Y1 - Y2$$
$$a = X2 - X1$$

$$L = \frac{Y1 - X1}{X2 - X1}$$

Die einzelnen Werte für X ergeben sich aus dem Anfangswert X1 + S, welches von 0 bis (X2 – X1) läuft. Die einzelnen Werte für Y ergeben sich aus Y1 – S multipliziert mit der Steigung L (siehe Zeile 70 und 80 im Programm).

Das Beispielprogramm arbeitet für die Werte X1 = 50, Y1 = 100 und X2 = 100, Y2 = 50.

Bei Änderung der Werte müssen die Formeln in Zeile 70 und 80 entsprechenend abgeändert werden.

```
5 REM HIGRA 8 DEMO
10 INPUT"X1=WERT ";X1
20 INPUT"Y1=WERT ";Y1
30 INPUT"X2=WERT ";X2
40 INPUT"Y2=WERT ";Y2
42 POKE 53265,59
44 POKE 53272,25
45 FOR V=1024 TO 2023
46 POKE V,16
47 NEXT V
48 FOR T=8192 TO 16383
49 POKE T,0
50 NEXT T
55 L=(Y1-Y2)/(X2-X1)
60 FOR S=0 TO (X2-X1)
70 Y=L*Y1-S
80 X=S+X1
90 GOSUB 200
95 NEXT S
97 GOTO 300
200 YP=INT(Y/8)
210 XP=INT(X/8)
220 A1=(YP*40+XP)*8
230 AY=Y-8*YP+A1
240 AA=8192
250 R=X-8*XP
260 M=2↑(7-R)
270 I=PEEK(AY+AA)
280 POKE AY+AA,I OR M
290 RETURN
300 FOR T=1 TO 5000:NEXT T
310 POKE 53265,27:POKE 53272,21
320 END
```

Wenn z. B. Y2 größer als Y2 ist, muß S in Zeile 70 zum Wert Y1 hinzuaddiert werden. Wenn X2 kleiner X1 ist, muß auch hier X im Laufe der Berechnung kleiner werden.

Literaturverzeichnis und Quellennachweis

1. Kopien der Datenblätter aus dem C-64 Reverence Manual von Commodore
2. Transactor, Zeitschrift aus Canada
3. The PET-Paper, Zeitschrift für Commodore Besitzer aus USA
4. Programme für VC-20, Hofacker Verlag
5. ELCOMP Artikel von Hr. Dipl. Phys. Peter Kittel
6. Tips zum Petting von Walter Waldner
7. PET Petits von H. P. Goertz
8. Handbuch zum Drucker VC-1515
9. Handbuch zum Commodore 64 und Floppy Disk Station
10. ELCOMP-Beitrag von Helmut Holighaus, Geschäftsführer der ADCOMP Datensysteme GmbH in München
11. Programmierhandbuch für PET, Nr. 110 vom Hofacker Verlag
12. Commodore 64 intern – Ein Data Becker Buch

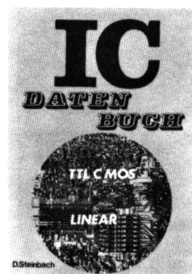
Hofacker-Bücher Deutsch



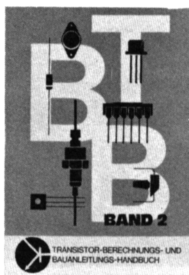
TBB-Transistor-, Berechnungs- und Bauleitungs-HB, Band 1, Hofacker
Völlig neu überarbeitete Auflage. Das Buch soll bei der täglichen Arbeit im Labor, in der Werkstatt oder am Elektronik-Hobbytisch Ihnen ein guter Begleiter sein. Berechnungsgrundlagen, Berechnungsbeispiele, Tabellen, Vergleichslisten, Digitaltechnik, Netzgeräte, BASIC-Programme zur Berechnung spezifischer Schaltungen usw. sind in übersichtlicher Form dargestellt. Ca. 300 Seiten.
Best.-Nr. 1 **29,80 DM**



Electronic im Auto, H. Gebauer
Mit Handbuch für Polizeiradar. Ein Buch für jeden technisch interessierten Autofahrer. Es zeigt Ihnen die vielen Möglichkeiten zur Verbesserung von Sicherheit, Leistung und Fahrkomfort in Ihrem Auto. Thyristorzündung, Beschleunigungsmesser, Drehzahlmesser, Batterieladegerät, Alarmanlagen u. v. a.
Best.-Nr. 3 **9,80 DM**



IC-Datenbuch, D. Steinbach
Daten- und Auswahllisten der gebräuchlichsten integrierten Schaltkreise. Digital und analog. Gerade bei ICs ist es wichtig die Anschlußfolgen genau zu kennen. Die wichtigsten TTL-Schaltkreise, NF-Verstärker, C-Mos Serie, lineare Schaltungen wie Operationsverstärker, Komparatoren, Spannungsregler, Trigger-Schaltungen, u. v. a. Das IC-Datenbuch wird auch Ihnen ein unentbehrlicher Begleiter bei allen Arbeiten mit integrierten Schaltungen sein.
Best.-Nr. 5 **9,80 DM**



TBB-Handbuch, Band 2, Hofacker
Dieses Buch ist die Fortsetzung des erfolgreichen Handbuches, Band 1. Ein Buch, das sich in der Hand des Praktikers bestens bewährt hat. Weitere neueste Schaltbeispiele und Berechnungsgrundl., Experimentier- und Versuchsbeschreibungen. Integrierte Spannungsregler, Wärmeableitung, Operationsverstärker Einführung, RC-Zeitglieder, Transistortester u. v. a.
Best.-Nr. 2 **19,80 DM**



IC-Handbuch, C. Lorenz
Ein Handbuch für digitale und lineare integrierte Schaltungen. Daten- und Auswahl-, Vergleichslisten, Gehäuseformen, Grundlagen, viele Schaltbeispiele, Printvorlagen, u. v. a. Alles über TTL-Technik, C MOS, MOS-Schaltungen, integrierte NF-Verstärker u. v. a.
Best.-Nr. 4 **19,80 DM**



IC-Schaltungen, D. Steinbach
Hier finden Sie eine gelungene Zusammenstellung der wichtigsten Anwendungsbeispiele aus dem Bereich der integrierten Schaltungen. TTL, C MOS, Linear. Alle Schaltungen sind übersichtlich und klar dargestellt und mit einer kurzen, jedoch sehr genauen Beschreibung versehen. Tastenenprellung, Zähler, Impulsgeber, Codierer, Dekodierer, Datenübertragung, Serien-Parallel-Wandler, Digitalvoltmeter u. v. a.
Best.-Nr. 6 **19,80 DM**



Elektronik Schaltungen, Hofacker
Die ideale Schaltungssammlung zum Basteln u. Experimentieren, Schaltungen mit Operationsverstärkern, Spannungsreglern, TTL, C-MOS-Schaltkreisen. MOS Uhr mit Wecker-elektronischer Würfel u. v. a.
Best.-Nr. 7 9,80 DM



IC Bauanleitungs Handbuch -IC -KIT, C. Lorenz
Ein Bauanleitungsbuch mit vielen hochinteressanten Bauanleitungen aus dem Bereich der LSI Schaltungstechnik. Schaltbeispiele mit Printvorlagen zum Selbstherstellen der Leiterplatten mit genauesten Beschreibungen. Hochaktuell und brandneu: Funktionsgenerator XR 2206, MOS-Uhr mit Wecker, programmierbarem Wecktongenerator, Schlummerautomatik, IC-Netzteil, Experimentieranleitung und Grundkurs über Flip Flops, u. v. a. Zu allen Schaltungen finden Sie Platinenvorlagen oder Sie können die Experimentierschaltungen auf der Experimentierplatine WH-1 g durchführen. Über 100 Seiten.
Best.-Nr. 8 19,80 DM



Feldeffekttransistoren, C. Lorenz
Der Feldeffekttransistor (FET) gehört heute zu den interessantesten Bauteilen überhaupt. Wie man damit experimentiert, wie man seine Funktion versteht und wie man damit brauchbare u. hochinteressante Schaltungen aufbauen kann, zeigt Ihnen dieses Buch. Grundlagen, Kennlinienfelder, Tabellen, Rechenbeispiele, Anschlußbilder und eine Vergleichsliste für Feldeffektransistoren bilden den Kern dieser umfangreichen Darstellung. Alles in allem finden Sie hier eine praxisnahe und komplette Arbietsunterlage, mit der Sie im Beruf und auch im Hobby erfolgreich arbeiten können.
Best.-Nr. 9 9,80 DM



Elektronik und Radio, C. Lorenz
4. Auflage. Völlig neu bearbeitet und stark erweitert. Eine Schritt für Schritt Einführung in die Radiotechnik mit vielen Bildern. Vom einfachen Diodenempfänger (Detektor) bis zu interessanten Sender- und Empfängerschaltungen (Minispione). IC-Radio, IC-Sender, Antennen, Berechnungsgrundlagen, Tabellen u. v. a. Über 150 Seiten.
Best.-Nr. 10 19,80 DM



NF-Verstärker, C. Lorenz
Grundlagen der integrierten NF-Verstärker, Berechnung von kompletten IC-NF-Verstärkerstufen. Anwendungsbeispiele mit den interessantesten und gebräuchlichsten Standard IC-NF-Verstärkern wie TBA 800, TBA 830, usw. Printvorlagen, Auswahltabellen, Experimentieranleitungen und Anschlußbilder machen dieses Buch zu einem unentbehrlichen Begleiter für alle, die sich m. NF-Verstärkern beschäftigen wollen.
Best.-Nr. 11 9,80 DM



BIS, Beispiele integrierter Schaltungen, H. Bernstein
Auf über 130 Seiten Anwendungsbeispiele mit integrierten Schaltkreisen, Zeitgeber 555, Funktionsgenerator ICL 8038, Opto Elektronik, Operationsverstärker, Festwertspeicher (ROM), u. v. a.
Best.-Nr. 12 19,80 DM



HEH, Hobby Elektronik Handbuch
C. Lorenz
Das Schaltungsbuch f. jeden Hobbyelektroniker, Schaltbeispiele und Bauanleitungen aus dem gesamten Hobbybereich. Lichtorgeln, Alarmanlagen, Eiswarngerät fürs Auto, PLL-Schaltungen u. v. a.
Best.-Nr. 13 9,80 DM



Viel mehr als 33 Programme für den Sinclair SPECTRUM, R. G. Hülsmann
Ein echt aufregendes Buch für jeden SPECTRUM-Besitzer. Viele Tricks und Tips. 33 Superprogramme wie 3D-Graphik, Crazy Kong, Musik-Computer. Unterprogramme in Maschinensprache, zehn kurze Progr. für den Spectrum mit 16K RAM wie Mondlandung, Spielautomat, Irrgarten, Todeshöhle usw. Sie werden begeistert sein!

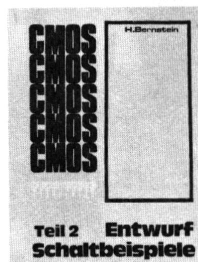
Best.-Nr. 144 29,80 DM



Opto-Handbuch, C. Lorenz
Das Handbuch für die gesamte Optoelektronik. Eine Einführung und ein ideales Nachschlagwerk. Grundlagen, Definitionen aller Kenngrößen, Opto-Lexikon, Berechnungsgrundlagen, Lichtsender, Lichtempfänger, Anzeigen, Infrarot Detektoren, Optokoppler, Opto-Vergleichsliste, u. v. a.106 Seiten.
Best.-Nr. 15 19,80 DM



C MOS Einführung, Entwurf, Schaltbeispiele, Teil 1 H. Bernstein
Vom C MOS Gatterbaustein über Schieberegister und Zähler bis hin zum C MOS Schreib- Lesespeicher. Insgesamt werden neunzehn interessante und bekannte C MOS Schaltkreise beschrieben. Zu jedem Bauelement sind genaue Daten, Schaltbild und Anwendungsbeispiele angegeben. Im großen Applikationsteil finden Sie: C MOS-Kippstufen, Addierwerke u. Rechenschaltungen, Digital Analog Wandler, Schieberegister für analoge Spannungen, Multiplexsysteme f. analoge Signale u. v. a. Eine komplette Einführung u. gut geeignet für das Selbststudium der C MOS Technik. 140 Seiten.
Best.-Nr. 16 19,80 DM



C MOS Entwurf u. Schaltbeispiele, Teil 2, H. Bernstein
Fortsetzung von Best-Nr. 16. Anwendungsbeispiele mit genauen Schaltungsbeschreibungen und Bauelementunterlagen. Daten, Anschlußbelegungen weiterer wichtiger hochintegrierter C MOS Elemente. Ein komplettes Arbeits- u. Experimentierbuch. C-MOS Uhrenschaltungen, Schieberegisterschaltungen, Parallel-Serien Umsetzung, statische u. dynamische Speicherschaltungen, Zählschaltungen, Digital Analog-Wandler, Analog Digital Wandler. Digital Voltmeter, I/O Registerschaltungen. RAM und ROM Anwendungen. Über 140 Seiten.
Best.-Nr. 17 19,80 DM



C MOS Entwurf u. Schaltbeispiele, Teil 3, H. Bernstein
Fortstetzung von Best.-Nr. 17. Eine sehr umfangreiche Applikationssammlung mit hochintegrierten C MOS Elementen. Speicher- und Steuerschaltungen, Multiplex- und Datenbussysteme, Liquid Cristal Anzeigen, Uhrenschaltungen, PLL-Schaltungen, Optoelektronik in Verbindung mit C MOS. Aufbau und Wirkungsweise der Prozeßrechentechnik, Arithmetische Logische Einheiten (ALU) u. andere wichtige Funktionen aus der Prozeßrechentechnik. RAMs, ROMs, und FIFO-Speicherschaltungen.
Best.-Nr. 18 19,80 DM



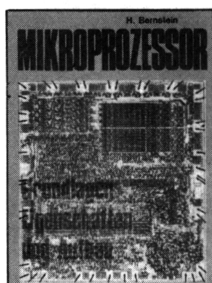
IC Experimentier Handbuch -IC, -EX, C. Lorenz
Eine sehr umfangreiche Schaltungs- und Bauanleitungssammlung mit neuen, jedoch meist beim Fachhandel erhältliche Standard ICs. Rechnerschaltungen, Mikroprozessoren, I/O-Schaltungen, Stoppuhren, druckende und anzeigende Rechner, Digitalvoltmeter, Hilfschaltungen für den Elektronik Experimentierer, A/D-Wandler, Frequenzzähler u. v. a. Viele Schaltungen können auf der IC KIT Experimentierplatine WH-1g aufgebaut werden.
Best.-Nr. 19 19,80 DM



Operationsverstärker, C. Lorenz
Dieses Buch umfaßt das gesamte Gebiet der linearen Schaltungstechnik und stellt ein in dieser Preislage bisher noch nie dagewesenes Nachschlagwerk und Einführungshandbuch dar. Bestens geeignet für das Selbststudium. Nach einer pädagogisch geschickt gemachten Einführung folgen theoretische Arbeitsunterlagen und die zugehörigen Schaltbeispiele mit Daten und Gehäuseanschlüssen. Dieses wertvolle Buch dürfte seinen Platz auch bei Ihren Arbeitsunterlagen finden, und wird dann immer von Nutzen sein, wenn es um die Lösung von nicht routinemäßigen Aufgaben geht. Über 150 Seiten.
Best.-Nr. 20 19,80 DM



Digitaltechnik Grundkurs, C. Lorenz
Ein Einführungskurs in die Digitaltechnik für Anfänger und Fortgeschrittene. Ein Fachbuch für den programmierten Selbstunterricht. Der ideale Kurzlehrgang für das Selbststudium. Der Kurs vermittelt Ihnen alle wichtigen Grundkenntnisse vom TTL-Gatter bis zum Mikroprozessor und Lösung von Schaltungsaufgaben durch Software. Viele Versuchsaufbauten u. Experimente aus diesem Kurs können auf der IC-KIT Platine WH–1g durchgeführt werden. Grundlagen, Gatter, Zähler, programmierbare Zähler, IC-Tester, Schieberegister, Speicher, Mikroprozessoren u. v. a.
Best.-Nr. 21 19,80 DM



Mikroprozessoren, Eigenschaften u. Aufbau, Teil 1, H. Bernstein
Grundlagen, Eigenschaften u. Aufbau von Mikroprozessoren. Organisation von Recheneinheiten und Mikroprogr. Programmierung und Klassifizierung v. Mikroprozessoren. Ablaufdiagramm, Flußdiagramm. Ein Cip-Technik und Multi Chip-Technik, Transfer- und Sprungfunktionen. Speichertechnik: RAMs ROMs, FIFO, FILO. Programmierbare logische Arrays (PLA). Anwendungsbeispiele u. Anwendungsbereiche. Über 120 Seiten.
Best.-Nr. 22 19,80 DM



Elektronik Grundkurs, C. Lorenz
Eine leichtverständliche und pädagogisch geschickt gemachte Einführung in die Technik der elektronischen Schaltungen. Ein Kurzlehrgang und Schnellkurs zugleich. Aber auch ein recht brauchbares Nachschlagwerk für den fortgeschrittenen Elektroniker. Mit wenig Mühe können Sie sich hier die Grundkenntnisse der elektronischen Schaltungspraxis aneignen. Das Buch schafft die Voraussetzungen für ein erfolgreiches und sicheres Arbeiten mit interessanten Schaltkreisen modernster Technologien. Unentbehrlich f. das Experimentieren mit den heutigen modernen hochintegrierten Schaltkreisen.
Best.-Nr. 23 9,80 DM

ohne Abbildung

Programmieren in Maschinensprache mit Z-80 – Band II, Dr. Schmitter
Dieses sehr interessante Werk geht insbesonders auf TRS-80 und Video Genie ein. Es kann jedoch grundsätzlich auf für alle anderen Personalcomputer, die auf der Z80 CPU basieren, verwendet werden (Color Genie, ZX-81, Spectrum, Osborne, Sharp, usw.). Aus dem Inhalt: μC, μP, RAM, ROM - eine Einführung, Unsere ersten Programmierschritte, Schleifen, Flaps und Sprünge, PUSH, POP, Stapel und Unterprogramme, Input-/Output-Programmierung und Cassettenport. Arithmetik, relative Sprünge und logische Verknüpfungen. BASIC und Maschinencode, wichtige Hilfsmittel wie Disassembler und Assembler.
Best.-Nr. 24 29,80 DM

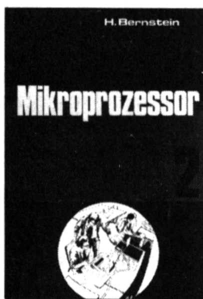
Achtung:
Unter Best.-Nr. 24 erschien bis vor kurzem das Buch Microcomputer-Technik von Blomeyer. Dieses ist vergriffen und wird nicht mehr neu aufgelegt.

ohne Abbildung

68000 Microcomputer Einführung
Eine leicht verständliche Anleitung zur Programmierung des leistungsfähigen 16 Bit Microcomputers von Motorola.
Erscheint ca. Mitte 1983.
Best.-Nr. 25 — 39,00 DM

Achtung:

Lieber Leser,
die Bestell-Nr. 25 und 27 waren früher Hobby Computer Handbuch (25) und Software Handbuch (27). Diese Titel sind vergriffen und werden vorerst nicht mehr neu aufgelegt.



Mikroprozessor, Teil 2, H. Bernstein
Die Fortsetzung von Best.-Nr. 22. Technologie von Mikroprozessor- und Speicherbausteinen. Festwertspeicher, PROM, EPROM, FIFO, Schieberegister, MPR-, ARL- und SAR-Register. Aufbau eines Mikroprozessorsystems mit 8080, RAM- und ROM-Schnittstellen. Befehlssatz 8080. Über 120 Seiten.
Best.-Nr. 26 — 19,80 DM



BASIC-M für Motorola EXORset®
Anwender Handbuch
Eine ausgezeichnete und professionelle Einführung in die BASIC-Programmiersprache. Ideal für die Industrie, aber auch ein ausgezeichnetes Werk für den Anfänger. Diese BASIC-Einführung mit vielen Beispielen gehört zu den besten deutschsprachigen Werken auf diesem Gebiet. Der Stoff geht auf den BASIC-M - Dialekt ein, ist aber auch auf jeder andere BASIC-Version anwendbar.
Best.-Nr. 27 — 29,80 DM



Microcomputer Lexikon u. Wörterbuch von A – Z, C. Lorenz
Einglisch/Deutsch – Der Fachausdruck wird übersetzt, ausführlich erklärt und erläutert. Deutsch/Englisch – Übersetzung des Fachausdrucks. Ein Hilfs- und Arbeitsbuch für jeden, der sich heute mit der modernsten Elektronik beschäftigt. Viele engl. Ausdrücke werden heute in der Elektronik, Computer- und Mikroprozessortechnik verwendet und oft fehlt uns eine genaueste und präzise Erläuterung. Ein Lexikon und Wörterbuch in einem einzigen Buch vereint.
Best.-Nr. 28 — 29,80 DM



Microcomputer Datenbuch - (englisch) – ELCOMP
Endlich es es da ! Viele unserer Leser haben seit langem auf dieses Buch gewartet. – Und es hat sich gelohnt. Auf über 800 Seiten haben wir hier für Sie die wichtigsten (fast alle) Bauelemente zusammengestellt (Daten, Anschlußbilder etc.), die sich in den heutigen Personalcomputern befinden. Wir haben alle wichtigen Systeme durchforstet und die Bauelemente herausgesucht. TTL, CMOS, Linear, Spannungsregler, CPU-Schaltkreise, 6502, Z80, 8080, 8085, 8086, 1802, 2650, Z8 u. v. a.
Best.-Nr. 29 — 49,– DM

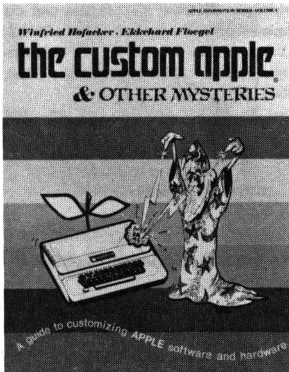
ohne Abbildung

Floppy Disk Selbstbau-Handbuch
von E. Flögel
Dieses Buch soll Ihnen auf einfachste Weise erklären, wie Sie Ihren Personalcomputer mit möglichst geringen Kosten mit einer Floppy Disk Station ersetzten können. Genaue Bauanleitung mit Software, Hinweise und Tips. Multiuser und Mutlitasking macht die Sache ganz besonders interessant.
Erscheint ca. Ende 1983
Best.-Nr. 30 — 49,00 DM

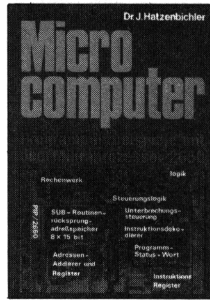
Achtung:
Unter Best.-Nr. 30 wurde früher ein Buch "Aktivtraining" angeboten. Dieses ist vergriffen und wird nicht mehr neu aufgelegt.



57 Programme in BASIC, Lorenz
Ein Buch mit techn.-wissenschaftlichen Programmen u. einer großen Anzahl von Spielprogrammen in BASIC (Games). Ein Buch für jeden, der sich mit dem faszinierenden Hobby der Mikrocomputertechnik befassen will. Alle Listings sind in BASIC und können auf den meisten Personal Computer Systemen gefahren werden.
Best.-Nr. 31 39,00 DM



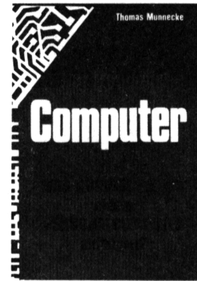
The Custom Apple & other Mysteries,
von W. Hofacker und E. Flögel
Dieses Buch bringt auf 190 Seiten Großformat eine Vielzahl von Erweiterungsprojekten für Ihren APPLE II. 6522 Ein-/Ausgabekarte, Tonerzeugung mit AY-3-8912, Analog/Digital-Wandler, EPROM-Burner, Anschluß des 8253 von Intel an den APPLE II, Schrittmotorsteuerungen mit APPLE II in BASIC, PASCAL und FORTH. Zu den einzelnen Projekten sind die notwendigen Platinenvorlagen sowie die Software im Buch enthalten. Alle Platinen zum Buch können vom Hofacker Verlag sofort ab Lager bezogen werden.
Best.-Nr. 680 (englisch) 79,00 DM



Mikrocomputer Programmierbeispiele für 2650, Dr. J. Hatzenbichler
Eine Einführung in die Programmierung von Mikrocomputern anhand des Prozessors 2650 von Signetics. Viele Programmierbeispiele in Maschinensprache, die Sie auf einem preiswerten Mikroprozessorsystem MIKIT 2650-P2 ausführen können. Zeitschleifenprogr., Blinkschaltung, Lauflicht, Stufenzähler, Stopuhr, Reaktionszeittester, u. a. Zu diesem Buch ist auch ein komplett aufgebautes und getestetes Mikrocomputersystem erhältlich, auf dem Sie alle beschriebenen Programme selbst ausführen können. Über 120 Seiten. (Nur solange Vorrat reicht, wird nicht mehr nachgedruckt.)
Best.-Nr. 33 19,80 DM



TINY BASIC Handbuch, Hermann
Das erste deutschsprachige Handbuch über Tom Pittman's TINY BASIC. Eine Einführung in die TINY BASIC-Programmiersprache. Wie kann ich meinen Computer (KIM-1) erweitern und BASIC programmieren. Systemvorschläge. Viele Programmierbeispiele, Tricks und Kniffe.
Best.-Nr. 34 19,80 DM



Der freundliche Computer – Was können Sie mit einem Personal Computer anfangen? T. Munnecke
Das Buch soll Ihnen auf die im Titel gestellte Frage eine ausführliche Antwort geben. Es eignet sich für alle, die bisher viel über Mikros gehört haben und gerne ausführlicher Bescheid wissen möchten. Viele interessante Fakten – Welche Computersprache ? Welche Anwendungen ? Welches Gerät soll ich mir kaufen ? 153 Seiten.
Best.-Nr. 35 29,80 DM

ohne Abbildung

Microcomputer und Roboter
Ein Buch für denjenigen, der sich externe Schaltungen für Microcomputer bauen möchte, die roboterartige Funktionen ausführen können. Spracherkennung, Analog-/Digital-Wandler, Digital-/Analog-Wandler, Ultraschallsensoren, Lichtschranken, Tonerzeugung u. v. a. Erscheint Anfang 1982.
Best.-Nr. 36 29,80 DM

ohne Abbildung

Oszillographen Handbuch
Ein Buch für jeden, der seinen Oszillographen optimal nutzen will. Der Anfänger, der noch nie einen Oszillographen benutzt hat, wird auf einfache Weise mit der Technik und Handhabung vertraut gemacht. Auch die Anwendung in Zusammenhang mit den modernen Mikrocomputersystemen wird beschrieben. Oszillograph als alphanumeres Darstellgerät u. v. mehr. Erscheint ca. Anfang 1982.
Best.-Nr. 103 19,80 DM

Z80 Assembler Handbuch
Dieses Handbuch braucht jeder, der ein Z80 Computersystem mit Z80 CPU hat (TRS-80, NEC, ALTOS, Sinclair, Video Genie usw.). Auf über 420 Seiten wird jeder Maschinenbefehl erklärt.
Best.-Nr. 8029 29,80 DM

Z-80 CPU Programming Reference Card
Eine praktische und kompakte Zusammenfassung der Z-80-Befehle, Z-80 Pinout, Registeraufbau, ASCII Zeichensatz kompletter Befehlssatz.
Best.-Nr. 252 5,00 DM

CMOS-Experimentierbuch
Eine kleine Einführung in die CMOS Schaltungstechnik. CMOS Grundlagen, Behandlungshinweise für CMOS Bausteine, Zusammenst. der wichtigsten CMOS Bausteine und deren Anschlußbilder. Viele praktische CMOS Schaltbeispiele. Ideal für jeden Elektroniker.
Best.-Nr. 106 1 5,00 DM

Handbuch für SC/MP
Ein echtes Handbuch für SC/MP (INS8060) Microcomputerbesitzer und solche die es werden wollen. Komplette Einführung mit vielen Schaltbeispielen, Schaltbildern und Programmlistings in Maschinensprache und TINY BASIC. Komplettes ELBUG-Listing, CPU-Karte, RAM-Karte, ROM-Karte, ROM-Programmierer, Cassetten-Interface, Fernsehinterface, Netzteil, Hex Ein-/Ausgabe, SC/MP Einkartenmicrocomputer u. v. a. 330 Seiten.
Best.-Nr. 108 29,80 DM

6502 Microcomputer Programmierung, P. Heuer
Eine deutschsprachige Einführung in die Maschinensprachenprogrammierung anhand des 6502 Microcomputers. Ein echtes Anleitungsbuch zum Einstieg in die Microcomputertechnik mit Hilfe des KIM-1. Viele Programmierbeispiele, die von einem Pädagogen speziell für Anfänger entwickelt wurden. Auch PET, AIM, SYM und ATARI-Besitzer brauchen dieses Buch.
Best.-Nr. 109 29,80 DM

Programmierhandbuch für PET
C. Lorenz
Ein für den PET produziertes Buch. Es beginnt da, wo Ihr mitgeliefertes Handbuch aufhört. Einführung Maschinenprogrammierung, Assembler, Ein-/Ausgabeprogrammierung, Programmiertricks, Analog / Digital-Wandler, Graphik, Spracherkennung u. v. a. mehr. Viele Listings, die Sie selbst eintippen können. Viele Informationen eignen sich auch für VC–20-Besitzer. 324 Seiten.
Best.-Nr. 110 29,80 DM



Programmieren mit TRS-80, Stübs
Das erste in einem deutschen Verlag produzierte Buch über den erfolgreichen Personal Computer von TANDY. Ein Buch für jeden, der einen TRS-80 bereits besitzt oder vor der Entscheidung steht, welchen Computer er sich anschaffen soll. Einführung, Programmiertricks, Erweiterungen, Maschinenprogrammierung und viele Programme (Listing mit Beschreibung). 202 Seiten.
Best.-Nr. 111 29,80 DM



PASCAL Handbuch, E. Flögel
Von BASIC zu PASCAL. Ein Einführungs- Lehr und Arbeitsbuch für jeden der sich mit PASCAL beschäftigen will oder muß. Viele Programmbeispiele, viele Tricks wie PEEK und POKE, Einbinden von Maschinenprogrammen u. v. a.
Best.-Nr. 112 29,80 DM



BASIC Programmierhandbuch
Einführung und Nachschlagewerk. Speziell für die BASIC-Versionen der modernen Microcomputersysteme. Jeder Befehl wird ausführlich beschrieben und ein Beispielprogramm gezeigt. Sehr übersichtlich und praktisch. Am Schluß finden Sie ein komplettes BASIC-Programm, das Ihnen über einen Computer BASIC lehrt.
Best.-Nr. 113 19,80 DM



16 Bit Microcomputer, J. Koller
Einführung, Daten, Eigenschaften, Anwendungen. Dieses Werk ist eine echte Sensation !
Alle 16 Bit Prozessoren werden beschrieben und erläutert. Applikationsbeispiele, Programmierhinweise. TMS 9900, 8086, Z8000, MC 68000, NS 16000, IAPX 486, IAPX 432. Über 370 Seiten.
Best.-Nr. 116 29,80 DM



Der Microcomputer im Kleinbetrieb
Das Buch für jeden Geschäftsmann. Auf über 170 Seiten erfahren Sie, was Sie als Gewerbetreibender oder freiberuflich Tätige über Microcomputer und die Anwendung wissen sollten. Geschichtlicher Hintergrund Geräteauswahl, Beispiele aus der Praxis, Programmbeispiele wie z. B. Textverarbeitung, Reisebüro, Ladenkasse, Adressverwaltung, u. v. a. Betriebswirtschaftliche Auswertung, Finanzbuchhaltung, Erfolgsanalyse mit dem Microcomputer, Liquiditätsrechnung, kurzfristige Erfolgsrechnung, Microcomputer für Freiberufler, Grundlagen der Finanzbuchhaltung für Microcomputeranwender. Dieses Buch kann Ihnen als Geschäftsmann für die Zukunft tausende einsparen.
Best.-Nr. 114 39,80 DM

ohne Abbildung

FORTRAN für Heimcomputer
Einführung in die FORTRAN-Programmiersprache mit vielen Beispielen. Grundsätzliches über die verschiedenen Microcomputersysteme, die bereits mit FORTRAN-Compiler lieferbar sind. Allgemeine Übersicht, Tips und Hinweise.
Erscheint ca. Ende 1982 .
Best.-Nr. 117 19,80 DM



Programmieren in Maschinensprache mit 6502, E. Flögel, W. Hofacker
Das deutschsprachige Werk über 6502 Maschinenprogrammierung. Einführung, Grundlagen, Eigenschaften, Adressierungsarten, Befehlsarten. Wie entwickelt man ein 6502 Maschinenprogramm? Handassemblierung, viele Programmbeispiele mit genauen Angaben direkt zum Eingeben in den Apple II mit Adressangabe (keine blutleeren Beispiele ohne Adresse), Verwendung von Assemblern. AIM-Assembler, Disassembler, Relocator, 6522 VIA, 6520, Interrupt, Fehlersuche in Maschinenprogrammen, Maschinensprache, Programmiertricks. Spezielle Abschnitte für Maschinensprachenprogrammierung über PET, CBM 3000, CBM 4000 u. VC-20, ATARI 400/800, Apple II, AIM sowie Ohio Scientific Challenger. Dieses Buch sollte jeder 6502 Systemanwender besitzen. Ca. 240 Seiten.
Best.-Nr. 118 49,00 DM

Anwenderprogramme für TRS-80 von Martin Stübs
Ein Buch, voll mit interessanten Anwenderprogrammen für TRS-80 Level II 16K und Video Genie (teilweise Diskette u./od. Cassette). Hauptsächlich Programme für den Manager, Geschäftsmann, Klein- und Mittelbetrieb. Auch einige interessante Spiele sind enthalten. Terminkalender, Reservierungsprogramm für Omnibusunternehmen und Hotels, Textverarbeitung, usw.
Best.-Nr. 120 29,80 DM

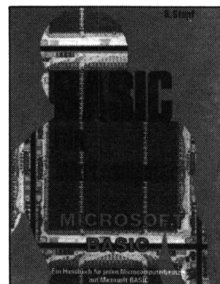


BASIC für Fortgeschrittene
Endlich ein BASIC-Buch für den fortgeschrittenen Programmierer. Alle wichtigen Befehle aus der Stringmanipulation, Disk-Befehle, WAIT, INSTR, WHILE WHEND usw. werden an Beispielen besprochen. Alle Befehle sind übersichtlich wie in einem Nachschlagwerk mit großen Überschriften angeordnet. Dann folgt ein umfangreicher BASIC Kurs für Fortgeschrittene mit vielen Beispielen (Inventur, Rechnungen schreiben, Adressenverwaltung usw.). Am Schluß finden Sie dann noch einen Vergleich der wichtigsten Sortiermethoden sowie ein Programm zur Vorhersage von Ereignissen.
Best.-Nr. 122 39,00 DM

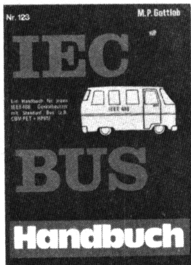




Programmieren in Maschinensprache (Z80), C. Lorenz
Eine sehr ausführliche Einführung in die Z80 Maschinensprache mit vielen Beispielen. Die Beispiele können mit Hilfe des TRS-80 Level II sowie dem T-BUG von TANDY und den T-BUG-Erweiterungen (IN LOCO, T-STEP, T-LEGS) ausgeführt werden. Ein unentbehrliches Buch für jeden, dem die BASIC-Programmiersprache von der Geschwindigkeit her zur Lösung seiner Aufgaben nicht mehr ausreicht.
Best.-Nr. 119 39,00 DM



Microsoft BASIC-Handbuch
Die deutsche Übersetzung des erfolgreichen Microsoft BASIC-Handbooks. Leicht verständliche Einführung mit vielen interessanten Programmbeispielen. Das kompetente Werk von Microsoft selbst. Ideal als Zusatzliteratur zu jedem BASIC-Buch.
Best.-Nr. 121 29,80 DM



IEC Bus-Handbuch, M. P. Gottlob
Ein Handbuch und Nachschlagewerk für alle Besitzer von Computern mit IEC (IEEE 488 Bus). Dazu gehört auch der PET sowie alle CBM-Computer. Grundlagen, das BUS-System, Meßdatenübertragung, Adressierung eines Instruments, kl. IEC-BUS-Lexikon u.v.a.
Best.-Nr. 123 19,80 DM



Programmierbeispiele für CBM
Ein Buch mit vielen BASIC-Programmen für CBM und PET. Spiele, Geschäftsbereich, Erziehung und Wissenschaft, Utilities, Hilfen für Maschinensprachenprogrammierung, trickreiche Programme. Viele Programme für wenig Geld.
Best.-Nr. 130 19,80 DM



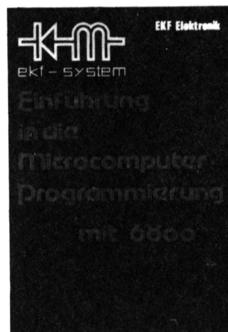
Programmieren in Maschinensprache mit CBM
An Hand eines praktischen Beispiels (Sortierroutine) wird der Unterschied zwischen BASIC und Maschinenprogrammen gezeigt. Das Maschinenprogramm kann mit dem leistungsfähigen MONJANA/1 Monitor in ROM erstellt werden. Am Schluß finden Sie weitere wichtige Informationen wie Dez/Hex-Umrechnungstabelle, Befehlslisten, ASCII-Tabelle sowie eine ROM-Vergleichsliste zwischen 8k PET und den neuen CBM-Maschinen.
Best.-Nr. 124 19,80 DM

MONJANA Monitor im ROM
Best.-Nr. 1241 79,00 DM

ohne Abbildung

ELCOMP
Fachzeitschrift
für Microcomputertechnik

Die kompetente Fachzeitschrift für das moderne Gebiet der Microcomputertechnik. Erscheint 2 x pro Jahr. Preis pro Heft 29,80 DM incl. MwSt., Porto und Verpackung. Wer die neuesten Informationen aus diesem **Gebiet** für sich nützen möchte, muß ELCOMP lesen. Software, Technische Tips, Programmiertricks, Bauanleitungen, Systembeschreibungen, u. v. a.
Best.-Nr. 125 29,80 DM



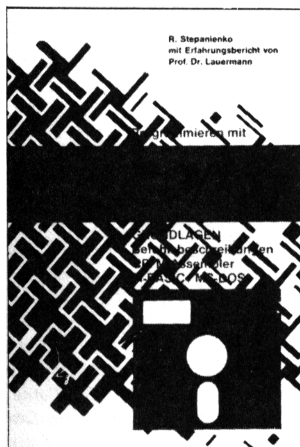
Einführung in die Microcomputer Programmierung mit 6800
Eine sehr gute Einführung in die Microcomputertechnik mit Hilfe des Mikroprozessors 6800. Ausführliche Erklärungen mit vielen Beispielen und Anleitungen. Theoretische Grundlagen. CPU-Architektur, Befehlssatz, Systemaufbau, Hilfsmittel der Programmierung, Trainingsprogramme, Systemkomponenten, FIRMWARE. Ein komplettes Monitorprogramm (Betriebssystem) ist als Listing enthalten. Über 250 Seiten.
Best.-Nr. 127 49,00 DM

ohne Abbildung

Programmieren mit dem CBM
Ein Hand- und Programmierbuch für alle CBM-Besitzer der 3000, 4000 sowie der 8000er Serie. Viele Tricks und Programmierbeispiele, Anleitungen.
Best.-Nr. 128 29,80 DM

ohne Abbildung

ELCOMP Leser Programmierhandb.
Hier fassen wir die besten Programme unserer ELCOMP-Leser zusammen. Programme für PET, CBM, TRS-80, AIM, Superboard, C4P, Exidy, Sharp, MZ80K, Apple II, Nascom I und II und TI 99/4 werden als Listing mit kurzer Beschreibung allen Lesern zugänglich gemacht. Erscheint Anfang 1982.
Best.-Nr. 129 69,00 DM



CP/M Handbuch

Grundlagen, Einführung, Hilfs- und Handbuch für jeden der mit dem "Software-Bus" arbeiten möchte. Ideal auch für Anfänger. Praktisches Handbuch für den Profi. Erscheint ca. Ende 1981.

Best.-Nr. 132 — 19,80 DM



FORTH Handbuch und Einführung von E. Flögel, W. Hofacker

FORTH ist nicht nur eine sehr leistungsfähige Programmiersprache – es ist schon fast eine "Religion". FORTH eignet sich bestens für industrielle Steuerungen, Grafik etc. Grundlagen und viele Programmbeispiele (Apple II, Ohio, ATARI usw.). Erscheint Mitte 1982.

Best.-Nr. 137 — 39,00 DM



BASIC für blutige Laien, Matthaei

Endlich ein Buch für den Anfänger und Laien. Ziel dies Buches ist es, dem "blutigen Laien" die Grundlage der Programmiersprache BASIC zu vermitteln. Lernen wird nun zum echten Vergnügen und Freizeitspaß. Auch der Preis macht Spaß.

Best.-Nr. 139 — nur 19,80 DM



Programmieren in BASIC und Maschinencode mit dem ZX81, E.Flögel

Ein Buch für Sinclair ZX81 Besitzer und solche die es werden wollen. Was heißt Programmieren?, Programmerstellung in BASIC, Spiele, Spielelemente, Programme für die Schule, Datenverwaltungsprogramme, Lagerbestand, Schallplattenverzeichnis, Programmieren in Z80 Maschinensprache, Anschluß einer PIO und externer Schaltungen, Lösen von digitalen Steuerungsaufgaben mit dem ZX81 u. v. a. Dieses Buch gehört auf den Tisch eines jeden ZX81 Besitzers (mehr als 20 komplette Programme, Maschinensprachen-Monitor, etc.).

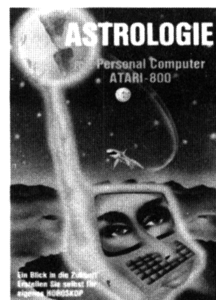
Best.-Nr. 140 — 29,80 DM



Programme für den VC-20 von W. Hofacker

Ein Buch des Hofacker Verlages speziell für den VC-20 Volkscomputer von Commodore. Viele komplette Programme wie Wortprozessor, Maschinensprachenmonitor, lustige Spiele, Programmieren in Maschinensprache, Ein-/Ausgabeprogrammierung. Wichtige Adressen des Betriebssystems, Tabellen, Speichererweiterungen, Dual-Joystick Bauanleitung u. v. a. Sie werden begeistert sein.

Best.-Nr. 141 — 29,80 DM

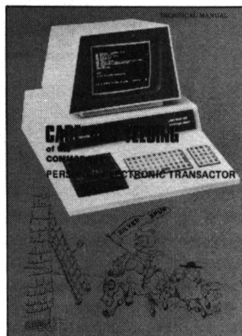


Astrologie mit dem Personal-Computer ATARI 800, W. Hofacker

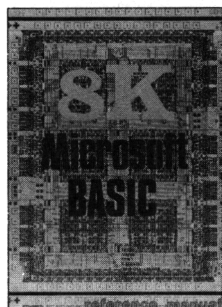
Ein Blick in die Zukunft. Erstellen Sie selbst Ihr eigenes Horroskop. Dieses Büchlein zeigt dem interessanten Leser wie man sein eigenes Horroskop mit professioneller Genauigkeit berechnen kann. Was braucht man dazu? Wie geht man vor? Was hat es mit der Astrologie auf sich? Was braucht man für die Deutung? Enthält ein komplettes Listing in BASIC und Ma.-Code (ATARI 800, 48K RAM + Disk).

Best.-Nr. 175 — 49,00 DM

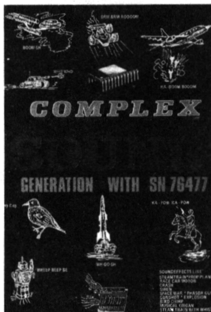
ELCOMP-Bücher Englisch



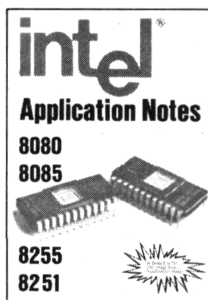
Care and Feeding of the Commodore PET
Das ideale Buch für den Hardware-Bastler. Viele Tricks, Schaltbilder, Hinweise und Erläuterungen für den, der gerne selbst Erweiterungen bauen möchte. Memory Map für 8k PET und CBM, Bauanleitung für eine serielle Schnittstelle u. v. a.
Best.-Nr. 150 19,80 DM



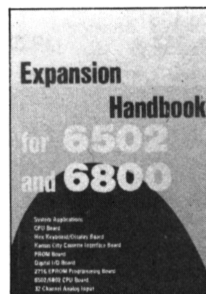
Microsoft 8k BASIC Reference Manual
Eine sehr gute BASIC-Einführung. Auch als Handbuch zum Nachschlagen bestens geeignet. Ideal für jeden PET, CBM, TRS-80, KIM-BASIC, SYM-BASIC, AIM- und APPLE-Besitzer. 73 Seiten DIN A4 mit vielen Beispielen.
Best.-Nr. 151 9,80 DM



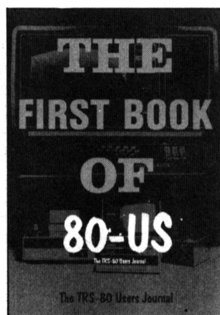
Complex Sound Generation with SN 76477
Ein Applikationsheft für einen der interessantesten integrierten Bausteine unserer Zeit. Ein LSI-Baustein zur Tonerzeugung. Je nach äußerer Beschaltung können Sie mit diesem Baustein die verrücktesten Töne erzeugen. Dampfeisenbahngeräusch mit Dampfpfeife, Vogelgezwitscher, Hundegebell, elektronische Orgel, Schuß mit Explosion u. v. a. mehr.
Best.-Nr. 154 9,80 DM



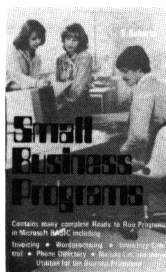
Expansion Handbook for 6502 and 6800
Das ideale Handbuch für alle KIM, SYM, AIM, PET und Challenger Computer-Freunde. Das Buch beschäftigt sich ausschließlich mit dem S-44-Bus. Dies ist exakt der Bus von SYM, AIM und KIM. Sehr viele Schaltbilder: CPU-Platine, Hex-Tastatur Eingabe, Knsas City Interface, RAM u. ROM-Karte, Analog-Eingabe Board u. v. a. Das Buch ist für jeden 6502 Systembesitzer unentbehrlich. Ca. 150 Seiten.
Best.-Nr. 152 19,80 DM



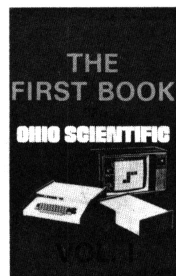
Intel Application Notes (8080, 8085, 8255, 8251)
Dieses Buch braucht jeder, der mit 8080, 8085 oder Z-80 Mikroprozessoren arbeitet.
Wir haben die interessantesten Applikationsberichte in diesem Buch zusammgefasst. Aus dem Inhalt: Designing with Intel's Static RAM's 2102, Memory Design with the Intel 2107B. 8255 Programmable Peripheral Interface Applications, Using the 8202 Dynamic RAM Controller u. v. a.
Best.-Nr. 153 29,80 DM



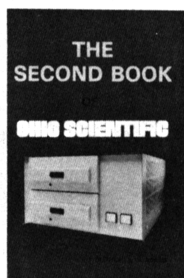
The First Book of 80-US
Für den TRS-80 Freund eine echte Preissensation. Die ersten fünf Hefte aus 80-US Journals in einem Sammelband zusammengefaßt. Voll mit vielen sehr interessanten Hard- und Softwareideen, Tricks. Viele komplette Programmbeispiele (Listings) in BASIC u. Z-80 Maschinensprache. Über 250 Seiten DIN A4. Farbiger Umschlag. Dieses Buch sollte jeder TRS-80 Besitzer oder der es werden will im Schrank haben.
Best.-Nr. 155 29,80 DM
(solange Vorrat reicht)



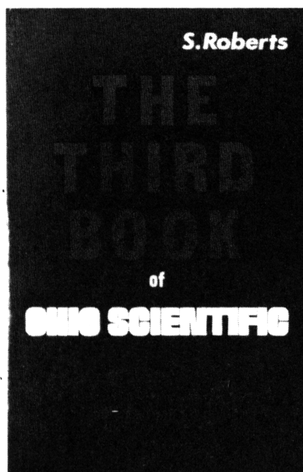
Small Business Programs, S. Roberts
Ein Buch für denjenigen, der die modernen Microcomputer (speziell TRS-80, Apple, PET, North Star, Challenger) zur Rationalisierung in seinem Klein- oder Mittelbetrieb einsetzen möchte. Viele nützliche Tips, Hinweise und Programmierbeispiele. Dieses Buch sollte jeder Geschäftsmann u. Microcomputerfreund besitzen.
Best.-Nr. 156 29,80 DM



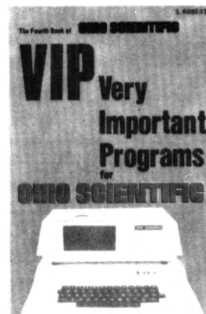
The first Book of Ohio Scientific
Das erste weltweit produzierte Buch für die erfolgreiche Ohio Scientific Challenger Computerserie. Grundlagen, viele Programmiertricks Hardwaretips, Umbauanleitungen, Programmierbeispiele u. v. a. Glanzumschlag, 186 Seiten.
Best.-Nr. 157 19,80 DM
(solange Vorrat reicht)



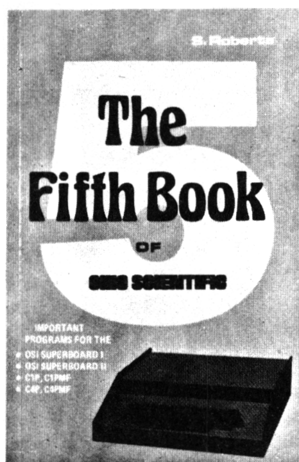
The second Book of Ohio Scientific
Eingehende Beschreibungen über praktische und geschäftsorientierte Software. Speicher Test Programm, Tricks und Tips für Disketten-Anwender. Mini-Floppy-Expansion u. v. a. 159 Seiten.
Best.-Nr. 158 19,80 DM



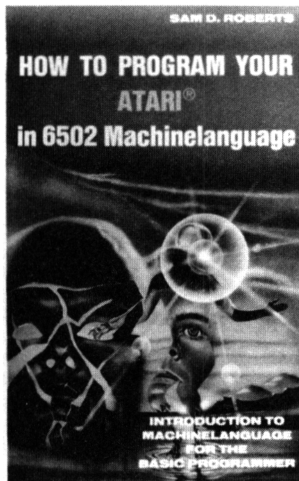
The third book of Ohio
Wie erweitere ich mein Challenger System ? Universelle I/O-Karte, EPROM-Burner für 2716, EPROM, RAM-Karte, 6522 VIA-Karte. Wo notwendig mit kompletter Software. Dieses Buch braucht jeder Ohio-Benutzer. Komplette Schaltbilder und Aufbauhinweise.
Best.-Nr. 159 19,80 DM

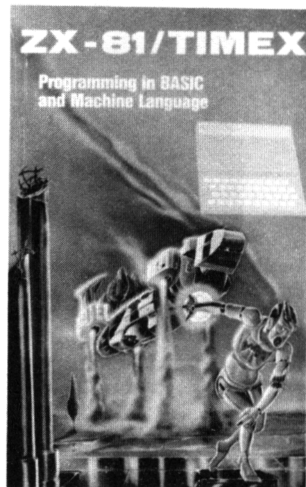


The fourth Book of Ohio Scientific
Ein Buch voll mit Programmen für das Superboard, C4P, C4PMF und C28P. Die Softwarequelle für jeden Challenger-Fan. Alle Programme sind getestet und auch auf Cassette verfügbar. 170 Seiten Listings und Beschreibungen.
Best.-Nr. 160 29,80 DM
Best.-Nr. 8324 Cassette 29,80 DM



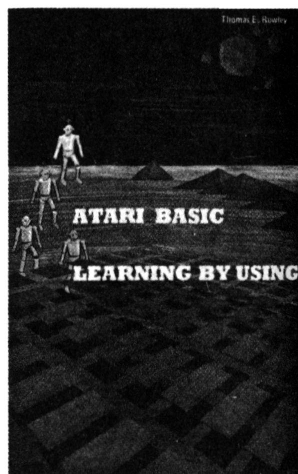
The Fifth Book of Ohio Scientific
Kein anderer Verlag der Welt hat fünf Bände für den so erfolgreichen und leistungsfähigen Personalcomputer bisher produziert. Der 5 Band bringt wieder eine Vielzahl von sehr interessanten BASIC- und 6502 Maschinenprogrammen. Neben einer Adressenverwaltung, einem Textprogramm, Fakturierprogrammen und Utilities finden Sie auch wieder viele interessante Spiele sowie eine komplette Abhandlung über Start- und Landesimulationen, die sich besonders als Hilfsmittel zur Entwicklung eigener Mondlandespiele eignen. Dem Buch kann vom erfahrenen BASIC-Programmierer auch viel Stoff zur Implementation f. andere Rechner entnommen werden (englisch).
Best.-Nr. 161 — 19,80 DM



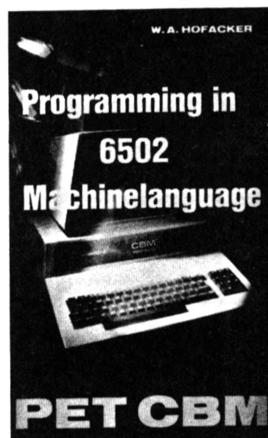


ZX-81 / TIMEX
Programming in BASIC and Machine Language by E. Flögel
Ein Buch mit vielen Tips und Programmen für den erfolgreichen Sinclair ZX-81 Personalcomputer. Dieses Buch ist eine Übersetzung unseres Titels Nr. 140.
Sehr interessant ist die Anleitung zum Aufbau einer Z-80 PIO an den ZX-81.
Best.-Nr. 174 — 29,80 DM

How to program your ATARI in 6502 Machinelanguage, Roberts
Eine sehr gute Einführung in die 6502 Maschinensprache mit dem ATARI 400/800.
Im einzelnen leicht verständlich und überschaubaren Lektionen werden Sie an Hand von praktischen Beispielen in die wunderbare Welt der Maschinensprache geführt. Sehr viele interessante Programmbeispiele für den ATARI. Tricks, Kniffe und Tips.
Best.-Nr. 169 — 29,80 DM



ATARI BASIC-Learning by USING v. Thomas E. Rowley (engl.)
Ein echtes Action-Buch für Ihren ATARI 400/800. Hier findet mehr statt als nur lesen. Sie verwenden es und machen neue Entdeckungen. Viele nützliche Routinen und• Hilfsprogramme. Grafik, Tonerzeugung, Joystickprogrammierung, PEEK und POKE und Special-"Struff".
Best.-Nr. 164 — 19,80 DM



Programming in 6502 Machinelanguage for PET + CBM
Dieses Buch enthält eine große Auswahl sehr wertvoller Informationen für den PET und CBM Besitzer der 3000er und 2000er Serie. Neben dem Listing und Beschreibung für einen sehr leistungsfähigen EDITOR/Assembler in Maschinensprache mit Beispiel für den Sie einen Disassembler, Linker, Editor, Assembler in BASIC und Maschinencode sowie einen kompletten Maschinensprachenmonitor für PET 2001. (engl.)
Best.-Nr. 166 — 49,00 DM

Hiermit gebe ich folgende Bestellung unter Anerkennung
Ihrer Lieferungs- und Zahlungsbedingungen auf.

ELCOMP

Menge	Artikelbezeichnung	Einzelpreis

Menge	Artikelbezeichnung	Einzelpreis

Unterschrift (für Jugendliche unter 18 J. der Erziehungsberechtigte) Datum



Hiermit bestelle ich die ELCOMP-Sonderhefte bis auf Widerruf.

○ Ich bitte um Abbuchung von DM 29,80 (incl. Versand und Verpackung) von meinem Konto (Giro oder Postscheckkonto)

Kto.-Nr. .... BLZ ....

Geldinstitut: .... Ort: ....

○ Ich bitte um Lieferung per Nachnahme. Mit der Lieferung eines Heftes wird der Betrag von DM 29,80 + DM 6,50 NN-Gebühr erhoben.

Name/Vorname

Straße/Haus-Nr.

PLZ Ort

Datum................ Unterschrift ....

ELCOMP

POSTKARTE



Absender
Bitte deutlich ausfüllen

____
Vorname/Name

____
Beruf

____
Straße/Nr.

____
Plz **Ort**

ELCOMP

MIKROCOMPUTER BOOK STORE

Tegernseerstr. 18

D-8150 Holzkirchen /Obb.

ABSENDER:

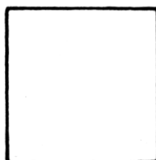
....
Name, Vorname

....
Straße

( ) ....
PLZ Ort

....
Telefon

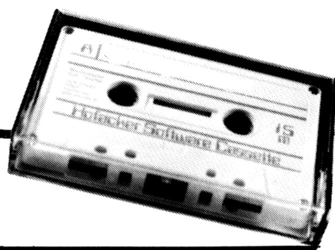
Die Bestellung der Sonderhefte kann JEDERZEIT widerrufen werden. Die Belieferung wird zum nächst erscheinenden Heft eingestellt.
Eine Kündigung NACH Auslieferung eines Sonderheftes (rückwirkend) ist NICHT möglich.



ELCOMP

Ing. W. Hofacker GmbH
Tegernseer Straße 18
D-8150 Holzkirchen

Leercassetten für Microcomputer



C–10

Die ideale Cassettenlänge für Ihren Personalcomputer.
Praktisch – handlich und betriebssicher

Kassetten mit nur 10 Minuten Spieldauer (2 x 5 Minuten) haben sich zur Aufzeichnung von Daten im Mikrocomputerbereich bestens bewährt.

Vorteile der C-10 Computer Cassette vom HOFACKER Verlag:

- weniger Bandsalat
- kurze Rückspulzeiten
- schnelles Auffinden von Programmen
- bessere Gleichlaufeigenschaften
- einfache Programmverwaltung

Die C-10 HOFACKER Datencassette bietet weiterhin:

- extrem hoch aussteuerbares Bandmaterial (Agfa)
- hochwertiges Cassettengehäuse, 5fach verschraubt
- Tefloneinlage für gute Laufruhe
- Staubdichtes Glasfenster

Die C-10 HOFACKER Datencassette wird seit 1978 speziell für Microcomputeranwender produziert. Die Cassetten bieten ein Höchstmaß an Betriebssicherheit bezüglich fehlerfreier Aufnahme und Wiedergabe.

Hier eine kurze Übersicht über die Anzahl der Bytes, die Sie auf eine C-10 Cassette abspeichern können:

Computer	Speichermöglichkeit	Computer	Speichermöglichkeit
ATARI 400/800	16K	APPLE	36K
Sharp MZ-80	32K	APPLE II	16K
AIM 65	16K	Heathkit	36K
Ohio Scientific	10K	Kansas City Std.	16K
TRS-80	16K	KIM-1	12K
TRS-80 Color Computer	24K	NASCOM	12K
Video Genie	16K	Exidy Sorcerer	12K
Sinclair ZX80/81	16K	SYM-1	12K

BESTELLSCHEIN

Menge	Beschreibung	Preis/DM	Gesamt
	1 Cassette	3,50	
	10 Cassetten	29,80	
	100 Cassetten	249,00	

Lieferanschrift

Name. . . . .
Straße . . . .
Ort . . . .

Datum . . . . Unterschrift . . . .

HOFACKER

Ing. W. Hofacker GmbH
Tegernseerstr. 18
D–8150 Holzkirchen
Tel.: (0 80 24) 73 31

Weitere interessante Bücher von Hofacker:

Bücher in deutscher Sprache aus dem Hofacker-Verlag

Best.-Nr.	Titel	Preis/DM
1	Transistor Berechnungs- und Bauanleitungsbuch – 1	29,80
2	Transistor Berechnungs- und Bauanleitungsbuch – 2	19,80
3	Elektronik im Auto	9,80
4	IC-Handbuch, TTL, CMOS, Linear	19,80
5	IC-Datenbuch, TTL, CMOS, Linear	9,80
6	IC-Schaltungen, TTL, CMOS, Linear (i. V.)	19,80
7	Elektronik Schaltungen (i. V.)	19,80
8	IC-Bauanleitungs-Handbuch	19,80
9	Feldeffekttransistoren (i. V.)	9,80
10	Elektronik und Radio	19,80
11	IC-NF Verstärker (i. V.)	9,80
12	Beispiele integrierter Schaltungen (BIS)	19,80
13	HEH, Hobby Elektronik Handbuch	9,80
15	Optoelektronik Handbuch	19,80
16	CMOS Teil 1, Einführung, Entwurf, Schaltbeispiele	19,80
17	CMOS Teil 2, Entwurf und Schaltbeispiele	19,80
18	CMOS Teil 3, Entwurf und Schaltbeispiele	19,80
19	IC-Experimentier Handbuch	19,80
20	Operationsvertärker	19,80
21	Digitaltechnik Grundkurs	19,80
22	Mikroprozessoren, Eigenschaften und Aufbau	19,80
23	Elektronik Grundkurs, Kurzlehrgang Elektronik	9,80
24	Progr. in Maschinensprache mit Z80, Band II	29,80
25	68000 Microcomputer Einführung (i. V.)	39,00
26	Mikroprozessor, Teil 2	19,80
27	BASIC-M Anwender-HB f. 6800/09/68000 (Motorola)	29,80
28	Lexikon + Wörterbuch f. Elektr. u. Mikroprozessor	29,80
29	Mikrocomputer Datenbuch (englisch)	49,00
30	Floppy Disk Selbstbau-Handbuch (i. V.)	49,00
31	57 Programme in BASIC	39,00
33	Microcomputer Programmierbeispiele	19,80
34	TINY-BASIC Handbuch	19,80
35	Der freundliche Computer	29,80
103	Oszillographen-Handbuch	19,80
108	Rund um den Spectrum (Progr., Tips u. Tricks)	29,80
109	6502 Microcomputer Programmierung	29,80
110	Programmierhandbuch für PET	29,80
111	Programmieren mit TRS-80 (Video Genie)	29,80
112	PASCAL-Programmier-Handbuch	29,80
113	BASIC-Programmier-Handbuch	19,80
114	Der Microcomputer im Kleinbetrieb	39,80
115	6809 Programmier Handbuch (i. V.)	49,00
116	Einführung 16-Bit Micorocomputer	29,80
117	FORTRAN für Heimcomputer	19,80
118	Programmieren in Maschinensprache mit dem 6502	49,00
119	Programmieren in Maschinensprache (Z80) Band I	39,00
120	Anwenderprogramme für TRS-80 u. Video Genie	29,80
121	Microsoft BASIC-Handbuch	29,80
122	BASIC für Fortgeschrittene	39,00
123	IEC-Bus Handbuch	19,80
124	Progr. i. Ma.-Spr. mit CBM, VC-20, C-64 (i. V.)	19,80
127	Einführung i. d. Microcmputer-Progr. mit 6800	49,00
128	Programmieren mit dem CBM	29,80
130	Programmierbeispiele für CBM	19,80
132	CP/M-Handbuch	19,80
133	Handbuch für MS/DOS (i. V.)	29,80
137	FORTH Handbuch	49,00
139	BASIC für blutige Laien	19,80
140	Progr. i. BASIC u. Maschinencode mit dem ZX81	29,80
141	Progr. für VC-20 (Spiele, Utilities, Erweiterungen)	29,80
143	35 Programme für den ZX81	29,80
144	33 Programme für den ZX-Spectrum	29,80
145	64 Programme für den Commodore 64 (i. V.)	39,00
146	Hardware-Erweiterungen für den C-64 (i. V.)	39,00
147	Beherrschen Sie Ihren Commodore 64	19,80
148	Programmierhandbuch für Sharp	49,00
149	Programme für TI 99/A (i. V.)	49,00
175	Astrologie auf dem ATARI 800	49,00
8029	Z-80 Assembler-Handbuch	29,80

Bücher in englischer Sprache

1. Von ELCOMP Publishing, Inc., Los Angeles, CA.

Best.-Nr.	Titel	Preis/DM
150	Care and Feeding of the Commodore PET	19,80
151	8K Microsoft BASIC Reference Manual	9,80
152	Expansion Handbook for 6502 and 6800	19,80
154	Complex Sound Generation using the SN76477	9,80
156	Small Business Programs	29,80
158	The Second Book of Ohio Scientific	19,80
159	The Third Book of Ohio Scientific	19,80
160	The Fourth Book of Ohio Scientific	29,80
161	The Fifth Book of Ohio Scientific	19,80
162	ATARI Games in BASIC	19,80
163	The Peripheral Handbook (i. V.)	29,80
164	ATARI-BASIC Learning by Using	19,80
166	Programming in 6502 Machinelanguage PET/CBM	49,00
169	How to Progr. your ATARI in 6502 Machinelanguage	29,80
170	FORTH on the ATARI – Learning by Using	29,80
171	See the Future with your ATARI (Astrology)	49,00
172	Hackerbook I (Tricks + Tips for your ATARI	29,80
173	PD-Program Descriptions (ATARI)	9,80
174	ZX-81/TIMEX Progr. i. BASIC a. Machine Lang.	29,80
176	Programs + Tricks for VIC's	29,80
177	CP/M – MBASIC and the OSBORNE (i. V.)	29,80

2. Von IJG Inc., Upland Californien

Best.-Nr.	Titel	Preis/DM
680	The Custom Apple & other Mysteries	79,00
681	Machine Language Disk I/O (TRS-80)	129,00

Riesenprogrammsammlung in BASIC

Best.-Nr.	Titel	Preis/DM
8048	BASIC Software Vol. VI	139,00
8049	BASIC Software Vol. VII	109,00
8050	BASIC Software Vol. I	69,00
8051	BASIC Software Vol. II	69,00
8052	BASIC Software Vol. III	104,00
8053	BASIC Software Vol. IV	29,00
8054	BASIC Software Vol. V	29,00

Der Hofacker Verlag produziert und vertreibt neben einer sehr großen Auswahl an Fachbüchern für Elektronik und Microcomputertechnik noch:

- Leerplatinen und Bauanleitungen für Zusatzeinrichtungen für Ihren Personalcomputer, sowie
- Programme (Software) für die bedeutenden Personalcomputer.

(i. V. bedeutet: Buch ist in Vorbereitung)

HOFACKER

HOLZKIRCHEN · SINGAPORE · LOS ANGELES

ISBN 3-88963-147-9

C. Lorenz

Beherrschen Sie Ihren Commodore-64

147